1998年11月1日発行　DOS/Vmagazine特別編集

SOFTBANK MOOK

Oh!X

復刊

不屈のパーソナルコンピューティング

DirectXプログラミング/Javaアプリケーション開発入門
ゲームデザインのすすめ/プロ，アマCG作家の競演
DreamcastはPCとの架け橋となるか？/その後のX68000
特別付録 CD-ROM2枚組

SOFT BANK
オー！エックス
定価2500円

POLYPHONY
DIGITAL

ソニー・コンピュータエンタテインメントが産み落とした、
次世代ゲーム開発拠点のポリフォニー・デジタルが人材を募集します。

アルゴリズムを考えるのが好きだ
アセンブラ・プログラマー
信号処理ならまかせろ
ゲームの企画は30以上持っている
SOFTIMAGEはバッチリだ
LightWaveなら自信あり
CG関連ツールを作ったことがある
MATHEMATICAがフェイバリット
1万ライン以上のCプログラムを書いたことがある
プログラムの命はクロック数だと思う
シナリオライティングで食べていける

CADCAMを使わせたらプロレベル
タイポグラフィーは大得意
グラフィックデザインの達人だ
プログラムは未経験だが、高等数学はOK
物理学者になろうと思っている
NURBSモデリングは慣れている
OpenGLを使いこなせるつもり
映像演出のプロ経験がある
小説を書いたことがある
コンピュータは知らないが、デッサン力はある
アニメの原画は得意科目

どれかにピクッときた人求む

プレイステーション用ソフト
グランツーリスモ 標準価格5,800円（税別）
好評発売中

■ 募集職種（PDS社員＊1）
1. アルゴリズム考案者
2. プログラマー
3. ゲーム企画プランナー
4. CGデザイナー
5. グラフィックデザイナー

■ 資格
30才以下
国籍不問

■ 仕事内容
先進的なゲームソフトの企画制作。

■ 待遇
契約社員（PDS社員＊1）
給与／経験、年齢、能力などを考慮のうえ、
当社規定による。
年俸制380万円（大卒）以上
+インセンティブ給与＊2
福利厚生／各種社会保険完備。交通費支給。
勤務地／東京（赤坂）
休日休暇／完全週休2日制（土・日）、
年末年始休暇など（年間130日）
勤務／スーパーフレックス制
（標準労働時間 9:30〜18:15 コアタイムなし）

■ 応募方法
1. 履歴書（写真貼付、希望職種明記）
2. 職務経歴書（書式自由）
3. 自己PR書（書式自由）
4. 作品（提出可能な方のみ）
※写真、ビデオ、MO、フロッピー、CD-R等OK
※返却を希望される方は、その旨を明記願います。
5. 返信用封筒
1~5を同封の上、下記宛先へご郵送下さい。
書類選考の上、面接日時等の詳細をご連絡します。

■ 宛先
〒107-0052 東京都港区赤坂8-5-26 赤坂DSビル
株式会社 ポリフォニー・デジタル「ゲーム制作希望」Oh!X係
問い合わせ先／03-5771-8111 採用担当

会社概要
会社名／株式会社 ポリフォニー・デジタル
設立／1998年4月2日
資本金／1000万円
［(株)ソニー・コンピュータエンタテインメント100％出資］
事業内容／コンピュータソフトウェアの企画、制作等
事業所／東京（赤坂）
代表作／モータートゥーン・グランプリシリーズ
グランツーリスモ

株式会社 ポリフォニー・デジタル
Sony Computer Entertainment Group

＊1 PDS: Product Development Staff ／ゲームソフト制作専門職の新しいスタイル　＊2 作品の売り上げに応じて支払われるボーナス

SOFTBANK MOOK

Oh!X

復刊記念号

C O N T

<スタッフ>

●編集/植木章夫 ●協力/有田隆也 石上達也 市川幹人 岡崎栄子 小笠原陽介 影山裕昭 如月 緑 清瀬栄介 霧雨 桑野雅彦 腰原仁志 古村 聡 柴田 淳 須藤芳政 丹 明彦 都築和彦 新美 嵐 野澤絵美 古籏一浩 御木德高 森川久志 八重垣那智 山口 正 吉田賢司 吉田幸一 由水 桂 横内威至 プロジェクトチームDoGA 満開製作所

●カメラ/蒲生晴夫 新井邦彦 ●校正/フィールドアップ

●イラスト/長岡秀星 山田晴久 高橋哲史 岡村直也

●編集長/来島 樹 ●編集人/稲葉敏夫 ●発行人/岡崎 眞

●DTPデザイン クニメディア株式会社(渡部貴行/久木千佳子/次良丸里美/児玉直人/小林香名子) ●印刷/クニメディア株式会社

SD-X Celeron CDXII

Direct3D

Visual Laboratory

PowerVR2

PS/2マウスコンバータ

ちびちびX

© 1998 Shuusei Nagaoka

E N T S

マジック・アイ

昔，エレクトロニクス時代の始まりの頃，マジックアイという新型のラジオが流行しました。ダイアルを調節していって，電波が同調すると，孔雀の羽模様のような大きな緑色の目玉が瞬きして，好みの局をしっかりと捕らえたことを教えてくれました。思えば，このときが，人間の眼と電子の眼が深夜お互いを見つめあった運命的な最初の出会いだったようです。
電子の眼はどんどん成長を続けて，私たちが忘れてしまったこと，面倒な計算，つかみどころのない思想，輪郭の曖昧なイメージまで，聞き手の聞き方に応じてその場でディスプレイに表示し，もっとしっかりした質問をしなさいと，いつも待っているようです。
どうもこの話は，ずっと以前からあったことのように思われます。そう思って，古代の預言者たちがしたように，深夜ひとりきりで砂漠の真ん中に寝転がってみると，我々と世界の過去も現在も，たぶん，この先に起こることもすべて知っている宇宙の彼方の，もうひとつの大きな緑色の目玉と向かい合っていることに気がつきます。

表紙　長岡秀星
長崎県生まれのポピュラーアート界トップアーティスト。1970年大阪万博住友館デザインに参加後渡米し，カーペンターズ，アースウィンド＆ファイアーなどのレコードジャケットを手がける。
1984年　迷宮のアンドローラ発表
1985年　筑波科学万博ポスター制作
1990年　長崎旅博覧会ポスター制作
　　　　NHK特集シルクロードポスター制作
1992年　F1ジャパングランプリポスター制作
　以後イラストレーション活動を休止，米国で映像作品制作に注力。このたび新たな映画作品の構想が完成に近づき，イラストレーション活動も再開。ニュー長岡秀星ワールドの展開が始まろうとしている。

■広告目次

Oh!X復刊を迎えて

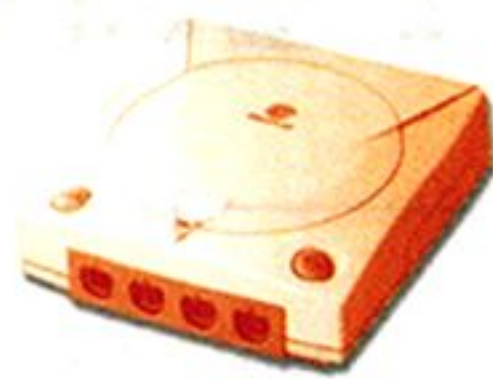
にんげんは昔から宇宙からの光が好きだった
だからいく時代も，日が暮れると草原に寝ころび，星降る空を眺めたものだろう。
ギリシャ神話にでてくるあの若者も，
そうやって宇宙海に広がる光の夢を織っていたのだ。
やがて，彼は金羊毛を探して船出した。目指すは未来……

MZ-80シリーズBASIC解説より

本棚の限られたスペースにいまも並ぶオレンジ色のマニュアル。私の本作りの基本の基本となっている１冊だ。これはパソコンが魔法の箱であった時代のBASICマニュアルで，そこにあるのは〝SCIENCE〟とその勝利。テクノルネッサンスの到来を宣言する書であった。
道具としてのパソコン，そしてそれを使いこなすという「遊び」。そうパソコンはそもそもホビーユースから始まったものだ。

当時，プログラミングは現在のPentium Ⅱ/450MHzマシンより高価なZ80/4MHzマシンを使っての高雅な趣味だった。そのマシンでなにができるかが問題ではなく，その遥か彼方に思いをはせつつパソコンを使っていたような気がする。単なる夢物語ではなく，その当時の環境とできることから，将来は簡単に定量化できたのだ。
CPUの演算能力が，画像表示能力が，音楽演奏能力が，記憶容量がそれぞれ桁違いに大きくなると，こんなことが実現できる……。
そして，現在では私たちは多くのものを手に入れた。それでどの程度幸せになれただろうか。
夢は実現できたのだろうか？　考えてみると得たものと失ったものがあることがわかる。
パソコンの処理はなぜか，高速だけど「重く」なった。非常に多くの公開された情報にも関わらず，パソコン自体へのアクセスは困難になった。
そうして，誰もがパソコンから一歩離れたところでつきあいをはじめていく。現在のパソコン業界を見ると，コアユーザーはすでに30代に移行し，広く普及しているわりには深くパソコンとつきあっている人はむしろ減ってきているといえる。踏み込んで使う必要がなくなったからというのは一面の真実であろう。だが，では今後は誰がパソコンを支えていくのだろうか。
現在，手に入れたと思っているものは本当に自分のものだろうか。失いつつあるものはないのだろうか。
そして本当に必要なものは，与えられるものではなく，すでに我々の身近なところにずっと存在していたのだということによようやく気づくことになる。
かつて「Oh!X」という名のパソコン誌で最後の青い鳥を我々は手放してしまった。それと同時に，ずっと培ってきた多くのものを失ってしまっている。
しかし，悲嘆するには及ばない。なぜ我々はOh!Xを手放したのか？　それが必要だったからだ。たとえ復活の見込みは薄くても，より大きな前進をするためには，機種別という鎖から解き放つことが必要だったのだ。
遅れてしまってすまなかった。
無為に時間を過ごしているつもりはなかったが，いろいろと手間取ってしまった。３年間でパソコン界はWindows95からWindows98に変わった。世間ではプレステが天下を取り，SATURNはDreamcastになろうとしている。

状況は大きく変わったか？
No.
OKだ。
遅れはしたが，遅すぎはしない。世間が変わっても我々は独自の道を行く。失ったものの回復には時間はかかるし，手間もかかる。しかし，それは十分に可能なことだ。周りを見渡すとずいぶん面白そうな玩具が手つかずで転がっているじゃないか。
Oh!Xの「X」はいうまでもなく，シャープX1のXから発祥している。このX1は，ご存じのとおり，テレビ事業部の実験機１号がそのまま名前になったものだ。その名のとおり，意欲的なパソコンだった。その気概をXの系譜は受け継いでいた。Oh!Xの「X」はいま一度，この「実験」へと立ち戻る。別にXをワイルドカードとして読んでもらってもかまわない。21.5世紀に向けて，さまざまな実験室としての意味を持たせてOh!Xは再発進を始めることにする。
まだ新しいOh!Xの礎は脆弱で頼りない。この先どこまで到達できるのかもまったくわからない。が，我々はとにかく前進していくしかない。まだ，やらなければならないことは山ほどあるのだから。

（U）

特別企画

Dreamcastの研究

Dreamcast

世間では話題になっているのいないのか……。もうすぐDreamcastがやってくる。1998年時点でのいわゆる「次世代ゲーム機」だ。

一般の人の反応がいまひとつ見えてこないのだが，ゲーム誌でDreamcast対応の新作が発表されても，内容的にはまったくインパクトのあるものではない。なにやらセガの自虐的なCMだけが話題になっている感もある。発売時期までにしっかり盛り上げてくれることに期待しよう。

Dreamcastのハードウェア仕様を見れば，そのポテンシャルはなかなかのものだということはすぐにわかる。少なくとも現状のゲームコンソールが3Dフィーチャーに関してはほとんどなにも備えていないという事実はそれをさらに大きな意味を持つものにする。Dreamcastは現状のゲーム機などはいうに及ばず，アーケードの最高システムをも上回る3Dグラフィック性能を提供する。ある意味で「ゲーム機だから」という甘えの通用しないレベルのハードウェアであるともいえる。そのトータル性能は，そう，ハイエンドPCに近づきつつある。PCの進化がゲームコンソールを超え，アーケードシステムを超え，新たななにかを実現しようとしている。そういった時代のゲームコンソールとしてDreamcastは実は重要な位置にあるのではないか。

WindowsCEの搭載，DirectXの採用，ネットワークへの接続。そして最高レベルのゲーム性能。PCの進む方向と対比すると，そこから導かれるものは意外と明解であるように思われるのだ。

PCの世界とクロスオーバーすればその位置づけは明確になるのだが，頑なに閉じた世界を構成するゲームコンソールの世界にあって，どこまで可能性を追求できるのか。最先端ハードウェアはゲームを変えることができるのか？　高機能端末としての可能性はどうなのか？

現時点での情報から推測される特徴をハードウェア中心に構成してみよう。そのうえで，そのコンセプトを見つめ直してみるべきだろう。

Dreamcastを分析してみる

Dreamcast™

中野修一

11月には発売されるというセガの次世代ゲームコンソール「Dreamcast」。さめた目で見ている人もいれば，かなり期待している人もいる。ちなみに，私はそれなりに期待しているほうだ。もちろん，発売直前にPlayStationの次世代機とかが発表されるとかいう展開もありがちだし，世の中Dreamcastだけで動くわけでもないのは確かだが，この年末の目玉商品であることには間違いないし，ここではDreamcastについてちょっと考えてみたい。

Dreamcastのひとつの特徴は，

公開されている情報がかなり多い

ということだ。CPUは日立のSH4，Dreamcast専用に作られたようなCPUだが，データシートなどは広く公開されている(これはSH2のときも同じだったが)。そして，グラフィック周りはNEC/VideoLogicのPowerVR2DC。これもPC用のチップが発売されるためか，それなりに情報が出てきている。そして，OSはマイクロソフトのWindowsCE。ゲームシステムはDirectX5というお馴染みのものが使われている。CD-ROMドライブとヤマハのサウンドチップだけはかなり特殊なものみたいだが，それ以外の情報はほとんどが公開されてるものばかりなのだ。

そういった面で推測できる部分が結構多くあり，Dreamcastグループを構成する各社はそれぞれの分野でトップレベルの製品を持ち寄ってきている。その結果，Dreamcastはなかなか侮れない力を秘めていることは容易に推測される。このようなものがコンシューマ市場に投入されるということはどういう意味を持つのだろうか。また，これだけのハードウェアを投入すればゲームというもののあり方を変えることができるのだろうか。我々に新しい地平を見せてくれるのだろうか。

現状のゲームコンソールに問題はあるか

かつて次世代機といわれたゲーム機も発売されてそろそろ4年が経過する。現状で勝ち残っているのはPlayStation(プレステ)だけだ。SATURNも市場としては成立しているが，社会的に見れば大きな勢力ではない。

かつては驚異的だったプレステのグラフィックももはやごく普通のものとなり，パソコンの能力も上がってきたので，クオリティという面では完全にトップレベルのものではなくなってきている。その代わり，普及台数はすさまじく，一時のファミコン，スーファミかそれ以上にプレステは広まってきている。真の意味で一般家庭にゲーム機が普及しようとしているのかもしれない。

絵が汚い，表現力が低い，CD-ROMが遅い，メモリが少ない，ロード/セーブが遅い……などといった我々が感じる不満点はおそらく一般の人はほとんど感じていないだろう。そもそも高性能機を出す必要があるのかというのが最初の疑問になってくるだろう。そんな必要はあるのか？　と。

それは実際にどれだけの変化が起こせるかによって変わってくる。性能向上というのは量的な変化にすぎない。しかし，それが一定量を超えれば質的な変化をともなってくる。たとえば，バーチャファイター(1)とバーチャファイター2の違いは結局は量的なものだが，質的な変化といっていいほどの違いが感じられる。それに対してバーチャファイター2とバーチャファイター3では，1から2になったときに匹敵するくらいのハードウェア性能の向上を認めたとしても，ゲームとしては量的な違いにしか感じられない。多くの人が求めているのは，ゲーム機の質的な変化である。

だから，「性能が上がる」という当たり前の事実はどうでもよいわけだ。問題は「十分に性能は上がるのか」という点になってくる。また，それを生かすだけのソフトウェアは制作できるのか，キラーアプリは出てくるのかといった点も重要だ。これまでにないものは出てくるのか，すなわち，質的な変化はあるのか。

もちろん，拡大しつつも停滞気味になっているゲーム機市場に新たな風が吹くことは悪いことではない。プレステのひとり勝ちにしておくのも今後を考えると問題があるのかもしれない。そのような状況のなかで見るとDreamcastというものがどれだけのことをやれるのかには興味深いものがある。まずはその辺をざっと見ていきたい。

Dreamcastのハードウェア

システム中枢から見てみよう。

●CPU

メインCPUは日立SH4が1個。SATURNはSH2の2CPU構成だった。ただし，性能が2倍になるという単純なものではなく，アセンブラレベルで精いっぱい使いこなしても，

「あまり幸せになれないです」

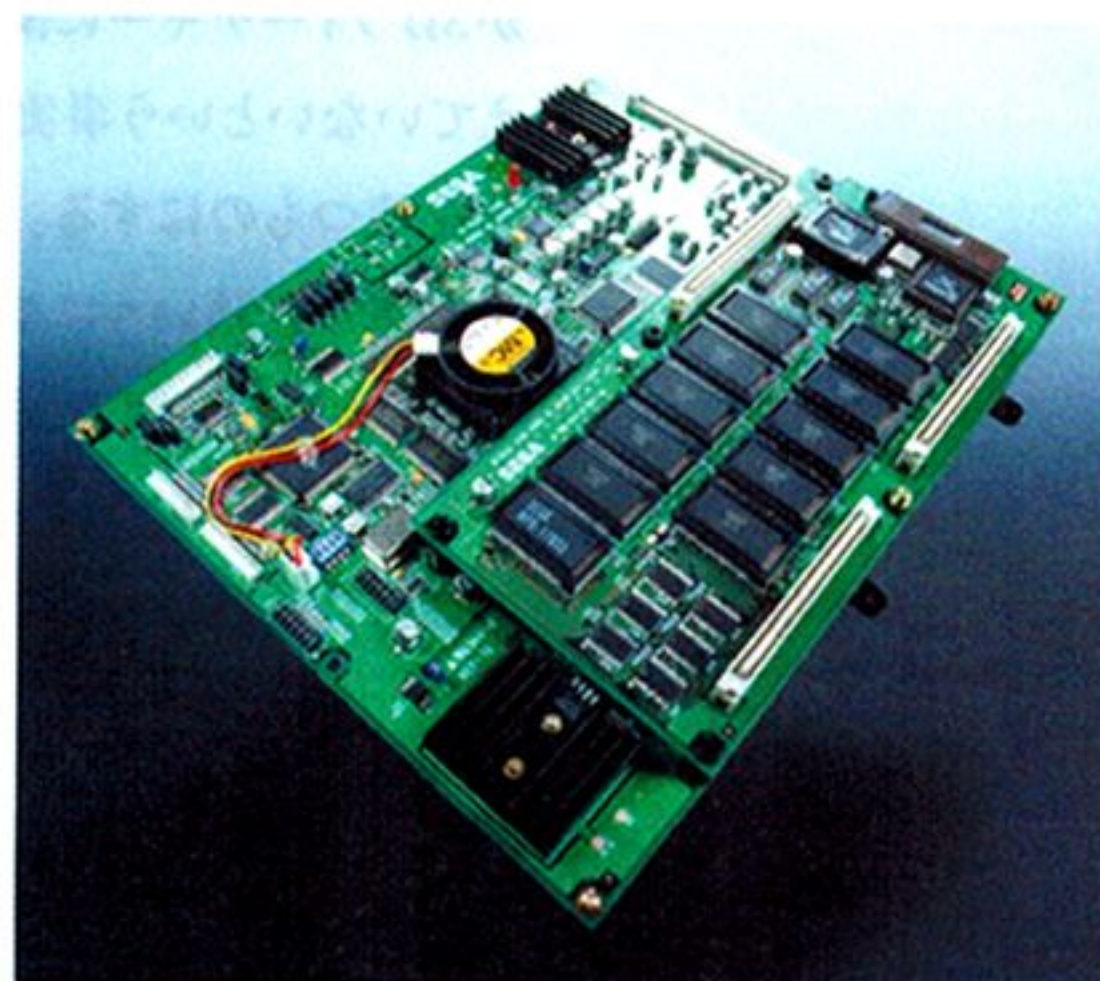
という話だったが，それでもSATURNの可能性をずいぶん広げたことは間違いない。

コストや使い勝手からいえば，1CPUで強力ならそれがいちばんだけど，当時最新鋭のSH2を採用していたのだから，それ以上を望むならマルチプロセッサという選択は間違いではない。SATURNの場合，つなぎ方とかがもっと違ってたらもっと展開も違っていたのかもしれないと思わせるものだった。

今回は十分に高速なCPUを搭載して1CPUで済ませようというわけだ。これもこれで間違いではない。最新高性能CPUだが，普通に買っても200MHz版SH4は1万個ロット時に4,000円ということで，意外と安く，大量に導入される場合には本体価格にさほど影響を与えずに済むだろう。SH2×2+SH1+68000という大量のCPUを内蔵していたSATURNだったが，十分に活用されていたとはいいがたい。多くのゲームではSH2を1個しか使っていなかったのではないかとも思われる。きっとSH4を1個のほうがたくさんの人を幸せにできるだろう。

このSH4は200MHzで動作する。周辺は100MHz程度で動作すると思われるが，SDRAMの限界から200MHz動作は期待できない。その代わり，バス幅は64ビットに拡大されているので100MHzメモリバスでも十分な帯域を持っていることになる。SH4の命令は2バイト固定長なので，1回のアクセスで4命令が取り込まれる。メモリ周り，キャッシュ周りとも練られているので，実際にかなりのパフォーマンスが発揮されると思われる。

さらに，浮動小数点演算ユニットに搭載された3D処理用のエンジンだが，1クロックで4つのデータを同時に処理できる。そしてアフィン変換や3次元変換を1命令で行ってしまう。その結果，1秒間に500万ポリゴン分の頂点演算ができるということになる。

大半の人にはすでに聞き覚えのある数字だと思うが，これがどういう意味を持っているか理解している人はどれくらいいるだろうか？　Dream

NAOMI基板。Dreamcastベースのアーケードシステムだ

castはDirect3Dでポリゴン処理を行う。Direct3Dでサポートされているポリゴン数は65536ポリゴンまでである。これ以上表示しようとしても表示できるかもしれないけど，なにが起きても知らないよ……とOtto Berkes(マイクロソフトの開発者)はいっていた。ということは，理論上限(65536×60fps)のポリゴンを演算しても79％のCPU負荷，多少は余力が残るくらいの力を持っていることになる。

今回のCD-ROMにはハイポリゴンカウントデータのサンプルとして，腰原仁志氏のLightwave 3Dで制作したキャラクターがなにも考えずにXファイルに変換したかたちで入っているが(細部を見るととんでもないデータだということはすぐにわかる)，それで25000ポリゴンくらいになる。だいたいのポテンシャルを計るにはよいサンプルになるかもしれない。

とにかく，これは現状でのジオメトリ性能としては破格のものだ。アーケード基板でもこうはいかない。現在のパソコンゲームはどんどん凄くなっているのだが，その大部分は3Dビデオカードによっている。しかし3DビデオカードはCPUの演算能力が足りなくて性能が発揮できていないというのはよくいわれることである。実際，最近のゲームはPentium II/400MHz以上と最新のビデオカードでないと真価を体験できない。素晴らしく美しく滑らかになってきた。複雑でリアルなモデリングを試みれば，それだけジオメトリ処理がネックになる。まず，そこからクリアしているというのは十分に評価できることだ。Dreamcastは，ポリゴンベースのハードウェアデザインとしては完成形に近いものになっていると思われる。

●メモリ

あい変わらず基本はSDRAMである。ただし，SATURNのときとは動作クロックなどが違う。現状ではこれ以上を求めるのは難しいだろう。

容量は16MBで64Mビットチップを2個使用する。現状の8倍というメモリ量はちょっと微妙なところだ。32MBあれば十分だろうと思うのだが……。

●グラフィック

ビデオチップとしてNEC/VideoLogicのPowerVR2が使われる。PowerVRはパソコン，コンシューマ機，アーケードゲーム機などをまとめて相手にしているものだが，これについては別項で詳しくやろう。なお，VRAM容量は8MBである。

●サウンド

ヤマハのスーパーインテリジェントサウンドプロセッサで32ビットRISCベース，DSP内蔵で3Dサウンド処理などを行うことができる。同時発音数は64音でAD PCMベース……かなり特殊なチップと思われるので，詳細についてはわからない。十分な性能であろうことは間違いないのだが。サウンド用のRAMは約2MB搭載されている。

●CD-ROMドライブ

1GBの倍密仕様が使える特別版で，最大12倍速と高速だ。大容量化はプロテクトの一環だろう。

●モデム

33.6kbpsのモデムを内蔵している。当然，ネットワーク対応のゲームが多数登場してくることが考えられる。ネットワークゲームはゲームの質的変化としてもっともとらえやすいものだが，日本でどこまで受け入れられるものか，また，ゲーム用のサーバはどうなるのか，料金は……といろいろ疑問も尽きない。最大のポイントであることは間違いない。11月末の本体発売時にこれらのゲームインフラが用意できるのか，どんなタイトルがネットワークゲームとして用意されているのかなどが注目される。8月末に某大手コンピュータメーカーの技術者がネットワーク要員としてセガに出向したとか聞いたのだが……間に合うのだろうか？

●ジョイパッド

かなり大型のものでどこかで見たような形をしている。7ボタンでアナログ仕様のスティックを備えている。スタートボタンを除いた6ボタンのうちの2つがアナログ仕様だ。アメリカのショウではデュアルショックのような振動ユニットがあったという報告もあったようだ。最大の特徴はなんといっても十字カーソルを採用しているということだろう。任天堂の特許が切れたか，任天堂絡みなのか，現在こちらでは未確認だ。

●メモリカード

ビジュアルメモリシステム(VMS)というらしいが，進化してWindowsNTになることはないのだろう(仮想記憶でもしてれば面白いんだけど……)。8ビットCPU内蔵の液晶付きメモリカードで単体でゲーム機になる。というよりは，やっぱりメモリカードというよりは，携帯ゲーム機だろう。すでに発売されているのでお持ちの方もいるかもしれない。128KBのメモリを搭載しているのは立派といえるだろう。画面は48×32ドット。単体で使えるのでいろいろ楽しそうだ。PDAとしての用途には多少の疑問は残る。ポケモンは偉大だったのだなあとつくづく思わされる。

ということで，現状で盛り込める仕様をすべて盛り込んだというハードウェア構成になっていることがわかると思う。これに対抗するハードウェアを現在の技術で作るのはかなり難しい。それくらいトップクラスのものばかりが集積されている。

もっともそれはSATURNのときも同じだったのだが……。

国産プロセッサSH4(7750)

船本昇竜@満開製作所

SH4の特徴を見る

SH4はその名前が示すように，第4世代のSHシリーズ。M68ファミリがそうであるように，基本的なアーキテクチャは，古い世代(SH2など)のそれとほとんど同じと考えてよい。例えるなら，SH2(7604)とSH4(7750)の関係は，M68030とM68060みたいなもの，といったところか。

しかし，M68シリーズのそれと大きく異なるのは，その方向性の違い。個人的な印象としては，SH2(7604)は「安く便利でなんにでも使えるやたら高性能なマイコン」というイメージに対し，SH4(7750)は「コンシューマゲーム機用プロセッサとして天下を取ったるでぇ」という鼻息まじり気味なイメージといったところか。

一般的に興味ある部分について

SH4(7750)は，いわゆるSHシリーズとオブジェクトコードレベルで上位互換性のある「32ビットRISCマイコン」である。ちょっとそそっかしい雑誌でよく見かける128ビットという数字は，グラフィックエンジンのそれを指す(編注：この表記だと68040は96ビットCPUということになる)。

動作周波数は200MHzで，カタログスペックでは，360MIPS，1.4GFLOPSを実現。ごく単純に動作クロックを比較した場合でも，SH2(7604)の28.7MHzの約7倍。加えて，SH4(7750)には，注目のジオメトリ演算機能，2つの命令を並列動作することのできるスーパースカラ，合計24Kバイト(SH2比6倍)のキャッシュ，さらに64ビットの外部データバスと，SH2(7604)に比べ，一桁違うパワーアップを実現する仕様となっている。

図1　処理モード別のCPUレジスタ構成

(a)ユーザーモードのレジスタ構成 (31〜0)	(b)特権モードのレジスタ構成(RB=1) (31〜0)	(c)特権モードのレジスタ構成(RB=0) (31〜0)
R0_BANK0	R0_BANK1	R0_BANK0
R1_BANK0	R1_BANK1	R1_BANK0
R2_BANK0	R2_BANK1	R2_BANK0
R3_BANK0	R3_BANK1	R3_BANK0
R4_BANK0	R4_BANK1	R4_BANK0
R5_BANK0	R5_BANK1	R5_BANK0
R6_BANK0	R6_BANK1	R6_BANK0
R7_BANK0	R7_BANK1	R7_BANK0
R8	R8	R8
R9	R9	R9
R10	R10	R10
R11	R11	R11
R12	R12	R12
R13	R13	R13
R14	R14	R14
R15	R15	R15
SR	SR	SR
	SSR	SSR
GBR	GBR	GBR
MACH	MACH	MACH
MACL	MACL	MACL
PR	PR	PR
	VBR	VBR
PC	PC	PC
	SPC	SPC
	SGR	SGR
	DBR	DBR
	R0_BANK0*1, *4	R0_BANK1*1, *3
	R1_BANK0*4	R1_BANK1*3
	R2_BANK0*4	R2_BANK1*3
	R3_BANK0*4	R3_BANK1*3
	R4_BANK0*4	R4_BANK1*3
	R5_BANK0*4	R5_BANK1*3
	R6_BANK0*4	R6_BANK1*3
	R7_BANK0*4	R7_BANK1*3

SHシリーズの基本構成

●汎用レジスタ (R0〜R15：32ビット×16本)

32ビットの汎用レジスタが16本で，最後のレジスタ(SH=R15，68K=A7)がハードウェアスタックポインタ(SP)になるので，一瞬，MC68Kを思い出させる。しかし，SH2のレジスタは，データとアドレスが区別されない汎用レジスタだ。また，ほかのRISCプロセッサでよく見られる，ゼロレジスタ(レジスタの内容がハードレベルで0に固定されている)はない。その代わり，「R0レジスタのみ有効」な命令が存在するなど，R0レジスタはアキュムレータのごとく，ひときわ使用頻度の高いレジスタとなっている。

余談かつ重要なことなのだが，実際にプログラミングを始めると，この実質15本のレジスタはなにかと足りなくなる。それも，微妙に(1〜3個)足りなくなることが多い。そのためか，SH4ではコスイ手法(レジスタバンクだもんな)ながら，レジスタが8つ増えたりする。

●コントロールレジスタ (SR，GBR，VBR，32ビット×3本)

CPUの状態などを表すステータスレジスタ(SR)は，Tビットや割り込み関連など，68Kなどとまったく同じシロモノと考えていい。ただし，SHにもSビットが存在するが，これは積和演算用ビットであり，スーパーバイザ状態ビットではないので，68Kユーザーは注意。

なお，SH2にはいわゆるユーザー/スーパーバイザの概念が存在しなかった。SH4は，ユーザーモードと特権モードの2種類の動作モードを持つようになった。例外もしくは割り込みで切り替わるそのモードにおけるM68のそれと大きく異なるのは，特権モード時には，なんとアクセスできる汎用レジスタの数が見かけ上8つ増え，24個になるということ。SH2経験者の「16個では足りない」の声が聞こえたのか。

グローバルベースレジスタ(GBR)は，GBR間接アドレッシングというアクセス方法を実現するための補助レジスタ。あくまでも，ベースアドレスの指定，という補助役なので，アドレスの代入以外の操作(汎用レジスタとGBR間の演算その他)はできない。

ベクタベースレジスタ(VBR)は，例外処理ベクタ領域のベースアドレスを示す。ハッキリいっ

て，システム基板のブートROMを記述するような人にしか縁がない。

　このように，コントロースレジスタ群は，一般のソフト屋がしょっちゅうアクセスするようなものではない(Tビットを除く)。ただ，GBRレジスタに関しては，若干，クセがあるものの，結構便利に使えたりもする。とはいっても，代用手段もクセもあるため，多用するプログラマとほとんど使わないプログラマに分かれるようだ。

●システムレジスタ(32ビット×4本)

　SH2(7604)の積和レジスタ(MACH，MACL)は掛け算(積和演算)用レジスタで，32ビットレジスタをペアで利用することで，64ビット値を扱うことができる。このレジスタのおかげ(?)で，32×32→64の演算が簡単にできるものの，その割には，使用頻度の高いXTRCT命令(ペアレジスタ64ビット中，中央32ビットを取り出す命令)が直接使えなかったり，パイプラインに結構長くいすわる命令に直接関与していたりするので，パイプラインをキツキツに詰めにくい(ある程度まで詰めるのは簡単)など，便利だけれども，注文も多いレジスタ。

　プログラムカウンタ(PC)は，ご存じのとおり，現在実行中のアドレスを指す。いま，これがないCPUはこの世に存在しない。

　やっかいなのが，プロシージャレジスタ(PR)。「サブルーチンプロシージャからの戻りアドレスを格納したレジスタ」とマニュアルにあるが，この厄介モノについてきちんと理解しているかどうかで「RISCアセンブラ」の経験があるかどうかがわかる。

●ロードストアアーキテクチャ

　SHはロードストアアーキテクチャ，つまりレジスタ間でしか基本演算が行えない。ワークメモリ内のデータを+1するだけでも，

```
    mov.l   work,r0  ;+1するアドレスを指定
    mov.l   @r0,r1   ;ワークの値呼び出し
    add     #1,r1    ;ワークの値を+1
    mov.l   r1,@r0   ;ワークに書き戻す
work:   .res.l  1    ;ワークメモリ
```

というように，文字どおり，(レジスタに)ロードして，(演算を行ってから，レジスタから)ストアするのである。i8085系(Z80など)を経験しているプログラマには懐かしい感覚かもしれない。

　ただし，ビット操作命令に関しては，ワークメモリに対して，アドレッシング方法が限定されているものの，直接実行できるようになっている。

アドレス空間について

　たとえばSHシリーズのSH2(7604)だと内部・外部データバス，汎用レジスタが32ビット(SH4だとバスは外部64ビット)。アドレス空間は，アーキテクチャ上は4GBと，M68020以上と同じ。というより典型的な32bitプロセッサのそれである。

　しかし，ここから先はほかのプロセッサと少し違っており，SHのマイコンとしての性格の強さを感じさせてくれる。

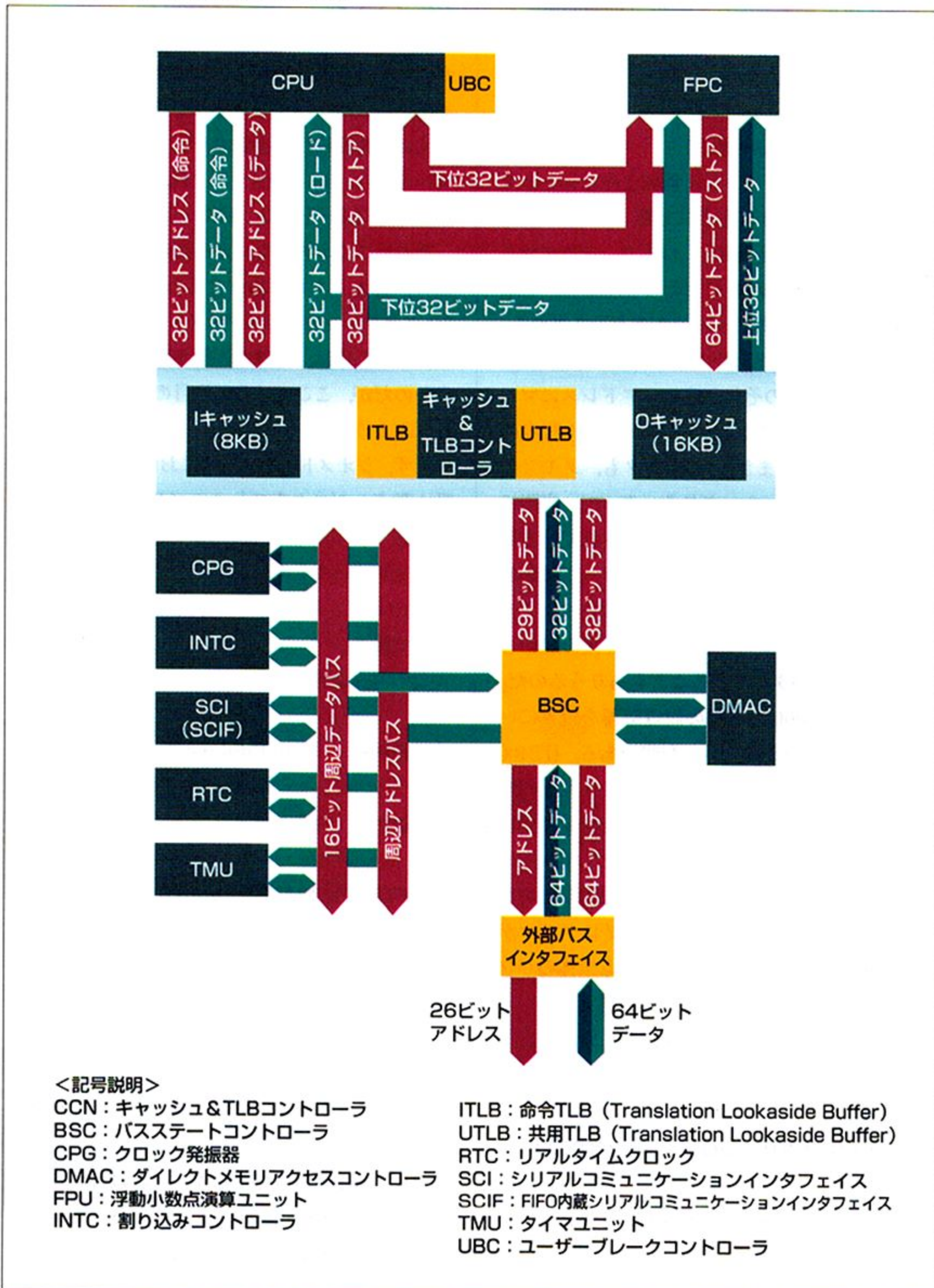
　まず，4GBのアドレス空間中，メモリ空間はリニア128MB。さらにそれらを，CS0～CS3の4つのリニアアドレス空間(32MB)として分割管理。加えて，それぞれに対してキャッシュスルー領域，連想パージ空間が存在する。こう書くと，古くからの「マシン語」プログラマは，バンクやセグメントといった単語を思い出すかもしれない。実際には，そんなに難しい問題ではないし，なにかと逃げ道も用意されている。

●CS空間

　まず，アドレスマップを見ても，アドレス空間は4つに分割されているとはいえ，メモリアドレスはあくまでも気持ちよいほどリニアで，最初は，なぜわざわざCS0～CS3というように分別管理する必要があるのだろうかと理解に苦しむほどだ。

　なぜか4つに分かれている空間の謎。実は，SH2などにはアドレス空間や制御信号を管理して

図2　SH7750機能ブロック図

<記号説明>
CCN：キャッシュ&TLBコントローラ
BSC：バスステートコントローラ
CPG：クロック発振器
DMAC：ダイレクトメモリアクセスコントローラ
FPU：浮動小数点演算ユニット
INTC：割り込みコントローラ
ITLB：命令TLB (Translation Lookaside Buffer)
UTLB：共用TLB (Translation Lookaside Buffer)
RTC：リアルタイムクロック
SCI：シリアルコミュニケーションインタフェイス
SCIF：FIFO内蔵シリアルコミュニケーションインタフェイス
TMU：タイマユニット
UBC：ユーザーブレークコントローラ

いるバスステートコントローラ(BSC)が搭載されており，これにより，CS0～CS3のそれぞれの空間に対して，メモリの種類，バス幅(8,16or32)やウェイト数などを個別に設定/管理することができるのだ。つまり，X68000でいうところの，メモリコントローラ(YUKIなど)が内蔵されている。よって，SH2やSH4のバスには，一般的には遅いとされるROMと，少し贅沢なSDRAMというような，複数の性質の異なるメモリを直接ぶら下げることができるのだ。

システムを起動するには，ブートROM(起動ソフト)とワークメモリが必要。といった基本的なシステムから，本当に速さが求められている部分にだけは高価で速いメモリを使う，というようなクセのあるシステムにも，ハードウェア上の複雑な設計――専用のメモリコントローラを別途用意――をすることなく実現できる。「部品点数が少ないのに便利」という便利なマイコンの王道を地でいっているといえよう。

●キャッシュスルー領域

アドレス上は，キャッシュ領域とキャッシュスルー領域というように完全に分離されているが，実際には同一の空間を指す。つまり，プログラム上でH'20000000番地をアクセスした場合，実際にはH'00000000番地のデータがアクセスされる。もちろん，CS0～CS3すべての空間が対応する。

通常，X680x0のそれと違い，アドレスにマッピングされた周辺機器(ジョイスティック程度のものから本当にさまざま)のデータも，メモリの場合と同様にキャッシュされる。よって，実際には，H'04000000番地にマッピングされている周辺機器の状態を見る場合，プログラムでH'04000000番地を読んだ場合，このデータはキャッシュされたもので，実際の(プログラムで読みにいった時点での)データと異なることもありうるのだ。よって，H'04000000番地の周辺機器を読みにいく場合は，キャッシュスルー空間である，H'24000000番地を読み込むといったプログラム上の配慮が必要である。もっとも，キャッシュを無効にするなどの方法もあるが，実際問題として，内部メモリのアクセス回数に対して，周辺機器のアクセス回数は圧倒的に少ないので，プログラマが「少し」配慮してやるだけで，システム全体のパフォーマンスアップにつながると私は考える。

たとえば，H'04000000番地にタイマがぶら下がっているシステムのプログラムを記述する場合，

```
TIMER_PORT:  .EQU     H'24000000
```

とし，タイマのアクセス時は常に，TIMER_PORTを使い，

```
mov.l     #TIMER_PORT,r0
```

というようにプログラミングすることで，キャッシュ云々を気にすることなく，正常にタイマを読めるようになる。プログラムの保守性も上がるので，一石二鳥だ。

ジオメトリ演算の実力

SH4にユーザー(それ以上にプログラマ？)が望んでいるポリゴン性能の強力なバックボーンとして備えられた，内蔵の浮動小数点コプロセッサおよび32本の32ビット浮動小数点レジスタ。5段パイプラインのそれは，モロ的にポリゴン指向で，レジスタセットや命令セットを見ていると，向こう側にポリゴンが見えるようで，こっちまでしみじみしてくる。

32本のレジスタはモードによってころころ名前が変わり，実際にレジスタは32本あるものの，基本的には16本が2セットあると考えるのが吉。たとえば，純粋に浮動小数点レジスタとして使用している場合，32本のレジスタは，FR0～FR31ではなく，FR0～FR15，XF0～XF15と呼ばれる。

それはさておき，なにかと話題になっているものの，実のトコロ正体がよくわからないものに，3Dグラフィック命令(ジオメトリック演算)がある。その正体は，特殊な浮動小数点演算と考えてよいのだが，ここではその注目の命令について触れてみよう。

まず，ジオメトリック演算においては，4個の浮動小数点レジスタをひとつのベクトルレジスタとして扱う。具体的にはFR0～FR3を1組にし，FV0というベクトルレジスタとする。また同様に，16個のレジスタを4×4の行列レジスタXMTRXとして扱う。

この前提条件で，

```
FIPR     FVm,FVn(m,n:0,4,8,12)
```

の1命令で，内積が求まり，

```
FTRV     XMTRX,FVn(n:0,4,8,12)
```

の1命令で，スケーリングができる。これが，ジオメトリ演算の正体だ。

……といっても，ポリゴンプログラム経験のない人には，スバラシクなんのことだかわからないだろうから，具体的に使用する局面の例を挙げてみよう。

まず，FIPR命令であるが，これはズバリ，ポリゴンの表/裏を判断するときに使う。たいていのポリゴンゲームには，こんなルールがあてはまる。「裏を向いているポリゴンは描画しなくてよい」。つまり，FIPRの1命令でポリゴンの表/裏を判断し，必要なポリゴンのみ描画を行えるのだ。

次に，FTRV命令だが，これもズバリ，ポリゴンの移動や角度の変更，さらには「拡大・縮小・回転」など(アフィン変換)，ありとあらゆる計算が一度にまとめてできる。これは，いわば四次元命令と呼ばれるもので，詳しく説明を始めると，ちょっとした数学になってしまうので省略するが，ポリゴン，つまり三次元座標をアレコレする場合は，ひとつ上の四次元(4×4：XMTRX)で計算をするのが常なのだ。

パイプラインとスーパースカラ

SH4(7750)は内蔵されたスーパースカラにより，ある一定の条件を満たせば，2つの連続した命令を並列実行することができる。その条件とは，連続する命令をうまく組み合わせること。たとえば，

```
and         r1,r0 ;EX
cmp/eq      #01,r0 ;MT
```

は，並列実行されるが，

```
sts         MACH,r0 ;CO
sts         MACL,r1 ;CO
```

は並列実行されない。もちろん，この並びには法則があり，次のように定義することができる。

まず，SH4(7750)の全命令を6つのグループに分類する。

MTグループ：比較(cmp/??)命令など
EXグループ：基本レジスタ演算命令
BRグループ：ブランチ(bsr)系命令
LSグループ：浮動小数点転送命令，メモリのロードストア命令など
FEグループ：浮動小数点演算命令など
COグループ：システム/コントロールレジスタ関連命令

そして，2つの命令グループの組み合わせにより，並列実行できるかどうかが分かれる。

第2命令

		MT	EX	BR	LS	FE	CO
	MT	○	○	○	○	○	×
第	EX	○	×	○	○	○	×
1	BR	○	○	×	○	○	×
命	LS	○	○	○	×	○	×
令	FE	○	○	○	○	×	×
	CO	×	×	×	×	×	×

ここで，例を振り返ってみよう。前者の同時実行可能な例は，EXグループのAND命令とMTグループのCMP/EQ命令の組み合わせ。後者の同時実行不可能な例は，COグループ命令の連続。なかなかユニークな機能なので，パイプライン詰めを極めたあとは，スーパースカラの効率的な運用を心がけるのが，正しい最適化道だろう。

さて，SH4(7750)のスーパースカラで特に注目したいのは「フロー依存関係時における，0サイクルレイテンシ(命令の発行とその結果完了)動作」。たとえば，次のような場合，

```
mov r0,r1 ;MT
add r2,r1 ;EX
```

MTとEXグループの組み合わせなので，これらの命令は並列実行できる。しかし，いざ同時に実行されたとしても，下のADD命令は，上のMOV命令の結果を使用するため，命令の完了を待たなくてはならない。……ハズなのだが，上の命令「MOV Rm,Rn」は0サイクルレイテンシ命令と呼ばれ，なんだか少し矛盾を感じなくもないが，命令のデコードが終了した時点で命令の実行が完了している，という特殊な命令なのだ。

パイプラインの流れを図示すると，次のようになる。

```
mov r0,r1 I D EX NA S
     ↓
add r2,r1 I D EX NA S
```

同時に開始された命令だが，上の命令は，「D」のステージで終了，つまり，R0の内容はR1へのコピーが完了しているので，下の命令の実行ステージ「EX」は，上の命令の完了を待つことなく，なにごともなかったように実行されるのである。おまけに0レイテンシ命令は，使用頻度の高い「MOV Rm,Rn」だけかと思いきや，ほかにも何種類か存在する。いったい，SH4(7750)の内部ロジックはどうなっているのだ？

合計24Kバイトのキャッシュ

SH4(7750)は命令用に8Kバイトの命令キャッシュ，データ用に16Kバイトのオペランドキャッシュが内蔵されている。SH2(7604)と同様に，オペランドキャッシュの半分のメモリ(8Kバイト)を内蔵RAMとして使用できるユニークなモードも健在である。

容量の増加にともなってか，SH2(7604)に比べ，細かな特徴の違いが出ている。まず，キャッシュ方式が4ウェイセットアソシエイティブ方式からダイレクトマップ方式になり，キャッシュラインサイズが32バイトと倍増。さらに，オペランドキャッシュは，コピーバック/ライトスルーの選択ができるようになった。動作周波数が200MHzにもなれば，「コピーバックはなきゃ話にナラン」というのが正直なところだが，ストアキューやプリフェッチ命令の装備など，ソフトウェアを意識した，キャッシュ周りの強化は評価できる。

64ビット外部データバス

SH2(7604)では4つに分割管理されていた物理アドレス空間は，SH4(7750)では，CS0～CS6の7つに分割して管理される。各エリアの最大サイズは64Mバイト。

また，SH2(7604)同様，バスステートコントローラ(BSC)を内蔵し，それぞれの空間に対して，各種メモリインターフェイスを持ち，バス幅を最大64ビットに設定できる。

また，PCMCIAインタフェイスも持ち，I/Oのバスサイジングもサポートしている。BSCの機能とは独立して，スマートメディアのインタフェイスも持っており，当初は，Dreamcastの記憶メディアはスマートメディアではないかと勝手に予想していたのだが，見事にハズれてしまったようだ。

そんなわけで，なにかと話題のDreamcastに搭載されるといわれるSH4(7750)のハデな部分を駆け足で見てきたが，どのような印象を持たれただろうか。

C O L U M N

SH Tips

●サブルーチンに気をつけろ

SH2のサブルーチン呼び出し命令である「BSR label」を実行すると、CPU内部では、戻りアドレス(「BSR label」の次の次の命令アドレス：遅延分岐命令なので、次の次のアドレスであることに注意)をPRレジスタに格納し、PCをlabelにセット(PCに、ディスプレースメントを加算)する。サブルーチン先で「RTS」命令に出合うと、PRに記録されている戻りアドレスをPCにセット(サブルーチンから復帰)する。

パッと見は、全然問題ないようだが、ここに落とし穴がある。確かにBSR命令は、復帰用に戻りアドレスを設定している。が、設定はPRレジスタのみ。つまり、「スタックにサブルーチンの戻りアドレスが積まれる」という従来のCISC CPUでは当たり前、かつ暗黙になされていた動作がないのだ。驚きの事実だが、最近のRISCは、たいていこのような仕組みになっている。そのような状態のまま、サブルーチン復帰命令RTSの前に、もう一度BSR命令がくる、つまりサブルーチンの多重呼び出しを行うと、見事にPRレジスタの内容はオーバーライトされ、最初の呼び出し元の復帰アドレスは消えてしまう(多重でない場合はメモリアクセスなしで行ける)。

それでは、SH2を含むRISCプロセッサは、サブルーチンの多重呼び出しができないのか？　と、そんなわけはない。ではどうするのかというと、ズバリ、プログラマが自分でスタック管理をすればいいのだ。とはいっても、正直いって自分でスタックを管理するのは、正直いって、面倒でミスも出やすい。また、保守性もよくないので、さっさと自分にあったマクロを作ってしまうのが得策だ。

●引数の受け渡し

上記のサブルーチンの例にもあるように、ハードウェアスタックポインタがR15に割り振られているものの、個々のCPU命令はハードウェアスタックをハナから無視しているフシがある。それがRISCというもの、と割り切って、各サブルーチンへの引き渡しは、レジスタで行うのが一般的だ。

```
                              ;**;命令実行後の様子
_main:  bsr     _tst1         ;02;PR=_end,PC=_tst1
        nop                   ;01;遅延分岐
_end:   bra     _end          ;11;ここには戻ってこない
        nop                   ;10;

_tst1:  and     r1,r1
        bsr     _tst2         ;04;ここでPRがオーバーライトさ
                                  れる
        nop                   ;03;
        rts                   ;09;PRがオーバーライトされてい
                                  るので、戻り先がわからない
        nop                   ;08;

_tst2:  and     r2,r2
        rts                   ;07;PRのアドレスに復帰(正常復帰)
        nop                   ;06;

        例：動かないbsrとrtsの組み合わせ


        sts.l   PR,@-sp       ;PRをストア
        bsr     _test1
        nop
        lds.l   @sp+,PR       ;ストアしたPRを戻す
        例：キチンと_test1にサブルーチン呼び出し


        .MACRO  _BSR          arg1
        sts.l   PR,@-sp
        bsr     \arg1
        nop
        lds.l   @sp+,PR
        .ENDM

        _BSR    _test1
        例：bsrサブルーチン用マクロ
```

Dreamcast™

WindowsCE搭載の意義

中野修一

WindowsCE搭載

Dreamcastは OSとして WindowsCEを採用している。ゲーム機にOS？　といぶかしく思う人もいるかもしれないが，最近のゲーム機はCPUも32ビットが当たり前だし，CD-ROMは当たり前だし（一部メーカーを除く），データのロード/セーブは必ずあるし，世界各国語に対応してるし……ということで，それなりにまとめてやったほうがいい処理というのは多くある。さらに今後はネットワークゲームへの流れも出てこようというところだ。その辺りをシステムがなにも用意してくれていなかったら，それはちょっと開発の敷居が高すぎる。

それでもWindowsCEといえばWindowsの縮小版だから，Win32APIを組み合わせてゲーム作るのかとか，馬鹿でかいMFCライブラリなんか使うのかと本気にしていない人も多くいると思う。起動時はWindowsCEでもゲームが起動したら，あとは全然関係ないのではないかというわけだ。

DirectXの採用

考えてみよう。Windowsのゲーム環境といえばDirectXだ。全面的に賞賛すべきものでもないかもしれないが，別にこれは使いものにならないものではない。よくこれだけ面倒なことをやってるなあと思うくらいいろんなことをやっている。Direct3Dなんて最初は開発者の誰も見向きもしていなかったものが，いまではリアルタイム3D環境のスタンダードだ。それだけのメリットがあり，開発ノウハウも確立されてきている。ということで，DirectX自体は別に不自然なものではないだろう。Dreamcastの場合，WindowsCEベースというよりもDirectXベースのシステムとして考えておくほうが違和感がないかもしれない。なにより，現状のWindowsCEはDirectXをサポートしていないので比較しようがないのだ。

当分の間，Dreamcastで使用されるのはDirectX5相当だが，やがてDirectX6相当に変わっていくという。とはいえ，DirectX6でサポートされる予定のバンプマッピングなどは最初からサポートされている。PCではDirectX7以降から対応が予定されているジオメトリ変換のハードウェア化にも対応している。CPU内部にジオメトリプロセッサを持ったマシン専用で出るのだからこのあたりは当然といえば当然だろう。

DirectXのライブラリ自体もPC版以上にチューニングされたものなので，よりパフォーマンスを発揮するという。PCの場合，Direct3DはDirectDrawを呼び出すかたちで実装されているが，同じチップが行うことならば一体化させたほうが都合がいい。3D表示のためにDirectDrawサーフェスを作る必要はないのだ。PCの場合は対応するハードウェアが確定できないので事情が違ってくる。このあたりも専用マシンの強みだろう。

ゲーム開発という面ではDirectX5用に作られたプログラム（ただし，Direct3Dはイミディエイトモードのみを使用）なら，ほとんどそのままDreamcastに持ってこれるという。Dreamcastの開発には特別なワークステーションなどは必要なく，VisualStudioベースでVisualC++中心で行われる。このあたりは既存のWindowsCE開発環境とほぼ同じと思っていいだろう。

ハード的に特殊である音源周り（さすがにMIDIデータで音楽を鳴らすわけではないらしい）と入力機器関係では多少修正が必要となるようだが，ライブラリ関数自体は上位互換仕様のような感じでは作ってあるので全面的な変更という事態にはならないだろう。とりあえず，音楽関係はDirectMusicではないということなのだが，今後DirectX6へ対応していくということなので，この辺りがどうなっていくのかにも注目しておきたい。音楽関係はPCとの互換性の最大の障害になる部分だと思われる。DirectMusicのβ版はDirectX6 SDKに収録されているので，パフォーマンスなどを検討してみるのもよいだろう。

さて，ライブラリレベルで互換なので，PC用のゲームが持ってきやすいというのは理解できたと思うが，ちょっと気になるのがメモリ量の違いだろう。昨今のPCでは32MBから64MB搭載するのが当たり前になってきている。こんな環境を前提として作られたゲームが16MBのメモリしかないDreamcastで動くものだろうか？

まず考えなければならないのはOSの大きさだ。Windows95の場合は起動時のメモリ使用量のうち20MB以上はOSで占められている。WindowsCEの場合，これは1MB程度にすぎない。さらにWindowsCEというのはメモリを圧縮して管理するシステムの上に成り立っている。これはDreamcastでも変わらない。パフォーマンスへの影響も気になるところだが，マイクロソフトによれば，実際にはなんら問題ないレベルだという（やや疑問も残るが……）。ここで使われる圧縮技術で約半分にデータ圧縮が行われるので，PC換算だと32～50MBメモリのマシンくらいだと思えばよいだろうか。さらに日立SuperHシリーズのコードの小ささを考慮すれば……とにかく，16MBといってもそう狭いメモリ空間でもないということは確かだ。

ついでに考えてみよう。なぜ，1MBでOSが収まっているのだろうか（普通に考えればそれでもデカイくらいだが，なんせWindowsだし）。WindowsCEのサイズは小さいとはいえ，DirectXが標準装備で加わっているのだ。

一般のWindowsCEマシンはWindows95ライクなGUIを備えているのだが，DreamcastではそのGUIをすべて取り払っており，ゲーム機用の軽いGUIを備えている。システムの内の主要なループはアセンブラで記述され，さまざまな軽量化を図っている。

DirectDrawは描画全般を受け持ち，DirectInputでさまざまなコントローラ（セガのことだから，たくさんの種類が出てくるのではないかと期待される）を制御し，DirectPlayでネットワーク対応のゲームが簡単に実現できるようになる。TCP/IPのサポートがされているのでインターネットへの接続も簡単だろう。PocketExplorerがついてくるという話は聞かないようだが。

その他の機能

DirectX以外の点でDreamcastがゲーム中でもWindowsCEマシンである部分としては，カーネルそのものが挙げられる。Dreamcastのカーネルはマルチスレッド，マルチプロセスに対応し，スレッド数に制限はない。WindowsCEそのものはシングルタスクOSなのは周知のとおりだが，内部はわりとちゃんとできているようだ。

CE自体のバージョンアップもありうる。すでに報じられているように，OSはCD-ROMに焼き込んで配布されるので，OSが変われば新しいのをつけてCD-ROMを作るだけでよい。その場合，古いゲームが新しいOSで動くようになるとかいったパソコン的な発想はしないほうがいいかもしれない。SATURNの例では，CD-ROMを開けるとリセットがかかるようになっていたので，新しいOSで起動してCD-ROMを入れ換えてほかのプログラムを動かすようなことはおそらく考えられていないだろう。たぶん，その必要性もほとんどない。OS自体がバージョンアップしてもユーザーには直接のメリットはないことになる。これではOSではなく，単なるアプリケーションランチャーなのだが，実際その性格が強いものであることは否めない。

そのほか，WindowsCEはメモリ管理，入出力

管理，ネットワークなどを担当する。普通のWindowsCE機（いわゆるHPC：Handheld PC）では外部記憶装置をパソコンに頼っている。パソコンと接続してデータをやり取りするというのが基本システムになっている。そういったパソコンとの接続なども考慮されているのかとか（疑問だ），WindowsCEとしての普通の機能の部分についてのほうがむしろ不明だ。

ゲーム以外の部分ではマルチメディアと通信関係を担当するらしい。単体でWebTVへ対応し，インターネット端末にもなる。キーボードはいまのところサポートされる予定はないようだが，ソフトウェアキーボードは標準でサポートされるので，一応たいていの処理はできるようにはなっている。

現在のセガサターンではキーボードが発売されているし，インターネット環境を考えると，キーボードは必須と考えてもよいものだと思われる。せっかくOS側の機能として日本語IME（インプットメソッドエディタ：かな漢字変換モジュールだと思えばいい）が装備されるのだから，むしろ用意されているほうが自然だろう。今後要望が多ければ対応もありうるということなので，発売されるのではないかと期待してもいいだろう（といって出てこなかった周辺機器も多いが……）。

GUIはないものの，WindowsCE2.1なので，SH3などのコードであれば既存のCE用のアプリは動くと考えてよいという。どうやってプログラムを転送するかとか，起動するかなどはわかっていないのだが。

Dreamcastと既存のWindowsCE用マシンとのもっとも大きな違いはDirectXがサポートされているかどうかだ。

今後はHPCでもDirectXがサポートされてくるので，次世代HPCとDreamcastとの接点も出てくるだろう。Dreamcastのハード的な特徴はそれぞれの機種のDirectX側で吸収できるだろうし，WindowsCEの製品はCPUごとにリコンパイルして出荷するのは当然なので，WindowsCEのシステムとして見れば単にオブジェクトが1種類増えるだけのことなのかもしれない。このあたりの互換性は微妙なところだ。

もちろん，これからのHPCがDreamcast並みの3D性能を持ったグラフィックチップを搭載してくるとは思えないので，おそらくは，Dreamcast用のプログラムをリコンパイルすれば一応は動くがとても遅い……という感じになるのではないかと思われる（甘いか？）。Voodoo2を前提にしたゲームをDirect3DのRGBドライバで動かすようなものだと思えばいいだろうか。ただ，マイクロソフトもDreamcastのDirectXが高度にカスタムチューンされたものであることは認めているので，互換性がなくても別に不思議ではない。

パソコンとの架け橋になるか？

これまでのゲームコンソールというのは閉じた世界であって，一般の人にはどうにも理解しがたく，情報もまったく入ってこないものだった。

PlayStationがある程度ユーザーにも開発環境を提供していたとはいえ，「ちょっと試しにやってみる……」というには高くつき，発表の場も限られていたので一般的なものにはなりえなかった。なにより情報の出版禁止条項などがびっしり決まっていて雑誌で取り上げることができなかったのだ。これは非常に面白くない。

Dreamcastの開発そのものはライセンスなどがいろいろ関わってくるだろうが，DirectX自体は公開された技術だ。WindowsCEの開発環境も公開されている。要するに，かなり近いところまでは開発の雰囲気なりを味わうこともできるだろうし，Direct3Dのイミディエイトモードを使うように気をつけていればDreamcastの開発環境とかなり近いところでゲームを作ることもできるわけだ。パソコンでDirect3D IMのプログラムを作っていれば，そのうちDreamcastに持っていく……といったこともありえなくはないかもしれない。

さらにPVRSG（PowerVR Second Generation：PowerVR2とも呼ばれている）自体はPC用としても発売される見通しなので，DirectX6でDirectX5相当の関数だけで作っていれば（バンプマッピングは可），ほとんど同じような環境でゲームを再現できることになる。

もっとも，ハイエンドPentium IIや3D Now!プロセッサを駆使しても，SH4に搭載されたジオメトリ変換エンジンをフルに使ったものと比べると見劣りがしてしまうかもしれない。しかし，それも来年早々に登場するというDirectX7に対応したジオメトリアクセラレータを使用することで一気に解決していく可能性もある。現状ではDreamcastの機能/性能はパソコンを大きく凌駕しているが，パソコンの進化というのは非常に速い。来年になればほぼ追いつくくらいのシステムが組めるようになると思われる（値段とOSの重さだけはどうしようもないが）。

これまで，パソコンとはかけ離れていたゲームコンソール。これからはただのWindowsパソコンがX68000以上にゲームコンソールに近い存在となるかもしれない。少なくとも開発環境はリニアにつながってくる。Dreamcastの登場はゲーム機市場以上にパソコンの世界に衝撃を与えるものなのかもしれないのだ。

なお，マイクロソフトはJupiterと呼ばれる新しいWindowsCEマシン仕様を決定している。年末にはJupiterマシンが現れてくるだろうが，WindowsCE2.1に対応したこのマシンがDirectXに対応しているかどうかはよくわかっていない。WindowsCE用のDirectXについては，いまひとつはっきりした情報がない。別に最悪，互換性は完全でなくてもかまわないので（互換性には期待していない），開発環境だけは同じになるようなものが早く登場してきてほしいと思うのは私だけではないだろう。

COLUMN

Direct3Dと演算精度

先日、AMD K6プロセッサの3D Now!との関連でDirect3Dに関する素朴な疑問を聞いてもらったのだが、意外と返事がこない。

すなわち、

「マイクロソフトによるとDirect3Dは倍精度（53ビット精度）を基準にして作成されているらしいが，演算を単精度でやって問題はないのか？　それとも倍精度でやっているのか？」

というものだ。3D Now!は32ビット単精度の演算を2個まとめてやるところにおいしさがあるので、おそらくは単精度演算で処理しているはずだ。ループを回して精度を上げることもできるのだが、それくらいなら素直にFPUを使ったほうがいいだろう。

DreamcastのCPUであるSH4でも32ビットデータを4つ同時に……というのがウリになっているので、おそらく単精度で処理していると思われる。Direct3Dを使っているならデータやパラメータなどでの互換性はあるはずだ。

DirectXの話なので、最初にマイクロソフトの広報から聞いてもらったのだが、返事はこない。Dreamcastのマネージャに聞いたところ、SH4は128ビットのグラフィックエンジンだから問題ないという答えだった。AMDのほうに聞いてもらったがこれもよくわからない。Meltdown Tokyoで米AMDの開発者に聞いたのだが、3D Now!では単精度演算を並列で行うものだということ以外に情報は出てこなかった。

簡単な質問だと思うのだが、質問の意図が誰にも理解してもらえていないらしい。「別に32ビットでも問題ないですよ」とひと言いってもらえれば納得するのだが……。

Dreamcast™

The PowerVR2 to make Dreamcast come true.

中野修一

今回のDreamcastでSH4のジオメトリ演算機能と対になる目玉がこのPowerVR2DCだ。これはNECとVideoLogicが共同で開発しているグラフィックアクセラレータだ。これまでは3D専用だったのだが，PowerVR Series2（PowerVR SG=Second Generationと呼ばれることもある。以下PowerVR2）では2D描画機能も加わった。PowerVRSGのうちゲームコンソール用のものをPowerVRDCと呼ぶようだ。PowerVR2自体は，PC用，アーケードゲームシステム用，ゲームコンソール用というラインアップになっている。具体的にPC用などとどこが異なるのかは不明だ。

PowerVR2は100MHzで動作し，300万ポリゴン/秒の描画性能を発揮するという。ちなみに，最新の3Dアクセラレータはどれもかなり凄い。下手するとアーケードシステムが太刀打ちできないくらいだ。ただ，性能の分，発熱ももの凄く，手で触れないほど熱くなるものも多い。PowerVR2もそれに劣らない性能を発揮するものだが，この手のチップでは初めて0.25μルールで製造されており，発熱はほぼ抑えられているという。家庭用にも安心なチップでしかも高性能に仕上がってきているようだ。

基本アーキテクチャ

●Tileベースアーキテクチャ

最近の3Dアクセラレータがどれも，エンジンを複数個並列で動作させることで性能を上げているのに対して，PowerVR2はTileベースで描画を進めていくアーキテクチャを採用している。

これはマイクロソフトのTarismanプロジェクトなどで研究されていた方式に近く，画面を32×16ドットの小領域に分割して，それぞれの領域ごとに描画を行っていくというものだ。512ドットが一度に処理されるわけではないだろうが，ハードウェアレベルでポリゴンのソートを行い（ドット単位），ハードウェアレベルで陰面消去される。外部のフレームバッファにはまとめてアクセスするという方式だ。

●Zバッファレス

PowerVRはもともとZバッファを持たないアーキテクチャとして知られていたのだが，PowerVR PCX2で32ビットZバッファを搭載してしまったので，話がややこしくなってしまっている。今度のPowerVR2ではZバッファを必要としない方式に戻された。

チップ外部のZバッファメモリとのやり取りではかなりのメモリアクセスが発生してしまう可能性がある。640×480ドット60fpsで32ビットZバッファを使用している場合，最低限で，

640×480×60×4×2

約140MB/sのメモリ帯域を必要とする。100MHz，64ビット接続のメモリ帯域は800MB/sだからまだ余裕はあるが，ダンジョンタイプのゲームなどではZへの上書きが多発し，300～500%のオーダーのZテストが発生していると見なしても別に不思議ではない。Zへのオーバーライトが行われるときはフレームバッファへのオーバーライトも同時に行われるので，帯域にほとんど余裕がなくなってくることもわかるだろう。これがスッパリとなくなるわけだから，かなり楽になる。少なくともZバッファを使用したシステムより何割か限界が高くなることがわかる。

●ディスプレイリストレンダリング

ディスプレイリストレンダリングが採用され，1シーンのデータを丸ごと転送して一括処理を行う。Tileベースで動作しているので，外部フレームバッファへのアクセスは完成画像の書き込みだけという最低限の理論値でフレームバッファへの負荷が減少していくことがわかる。

●Deferd Texturing

Deferd Texturingも効果的だ。これは実際に表示するピクセル分だけテクスチャメモリをアクセスする方法で，どんな複雑なシーンでも，画面ドット数以上にアクセス数が増えることはない（半透明とかは別か）。逆変換でテクスチャを求めて，Tileごとに完成させていくので無駄がない。シングルパイプラインでマルチテクスチャを楽にこなすという。

全体的に見て，外部フレームバッファへのアクセスを極端に減らすのがTileベースの利点だということがよくわかる。チップ内にタイル分の小容量高速メモリ（おそらくはオーバーサンプリング用でタイルサイズの4倍程度？）を持ち，そちらで処理を行ってフレームバッファに書き戻すということでメモリの負荷を減少させている。640×480ドットなら600個の領域に分けて処理を進めることになる。ディスプレイリストが重くなった場合の負荷の変化などがほかのアーキテクチャとは変わってくると思われる。

また，資料によれば32ビット浮動小数点ジオメトリセットアップエンジンを内蔵しているとある。SH4の機能とダブってしまっている感じはするのだが，3次元処理がハードウェアレベルでできるということを意味している。

その表現力

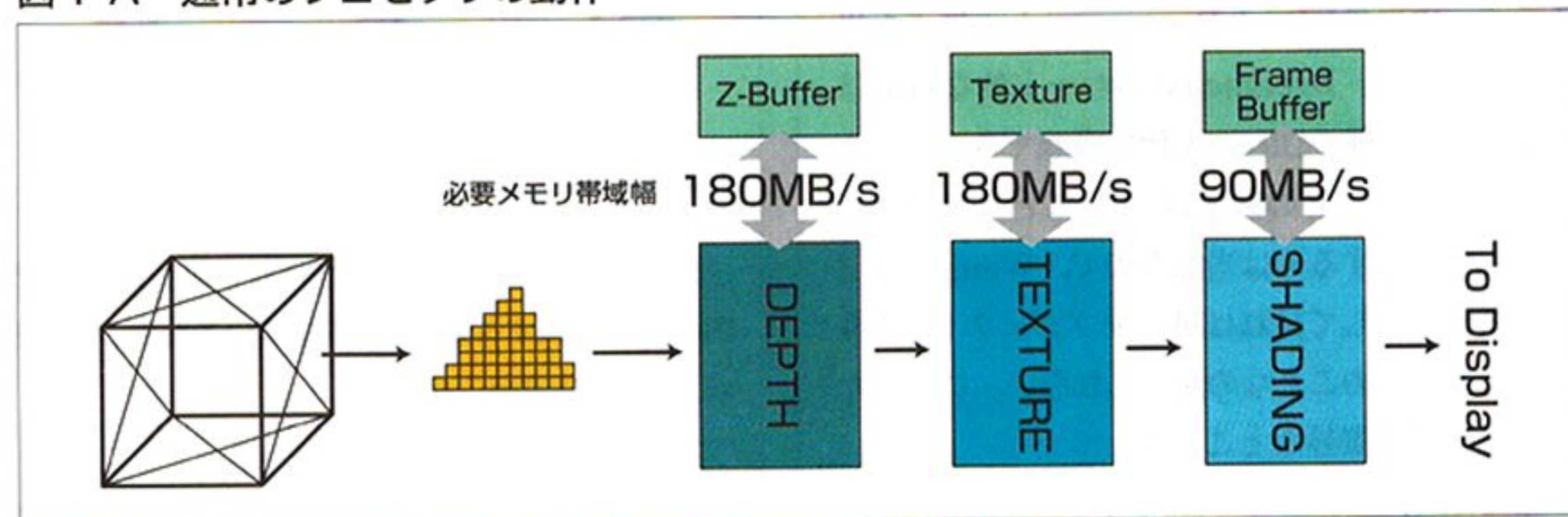
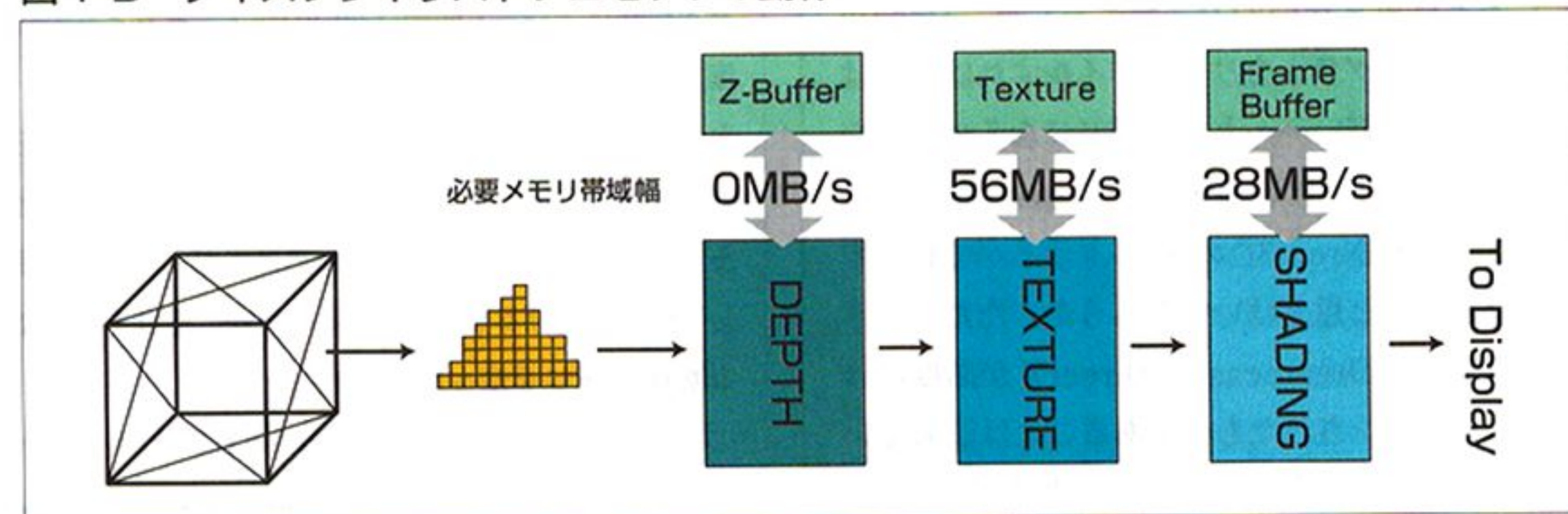
●バンプマッピング

テクスチャに凹凸を与える処理だ。最近の3D

図1-A　通常のプロセッサの動作

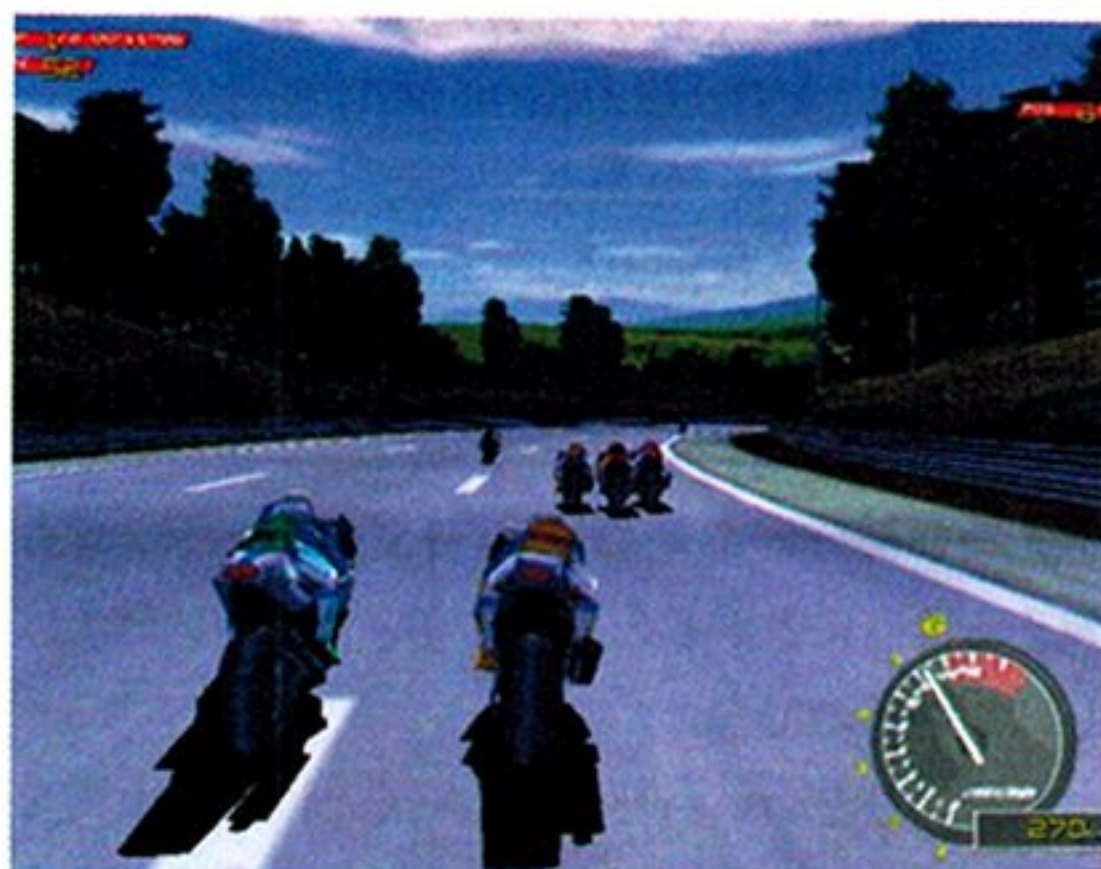
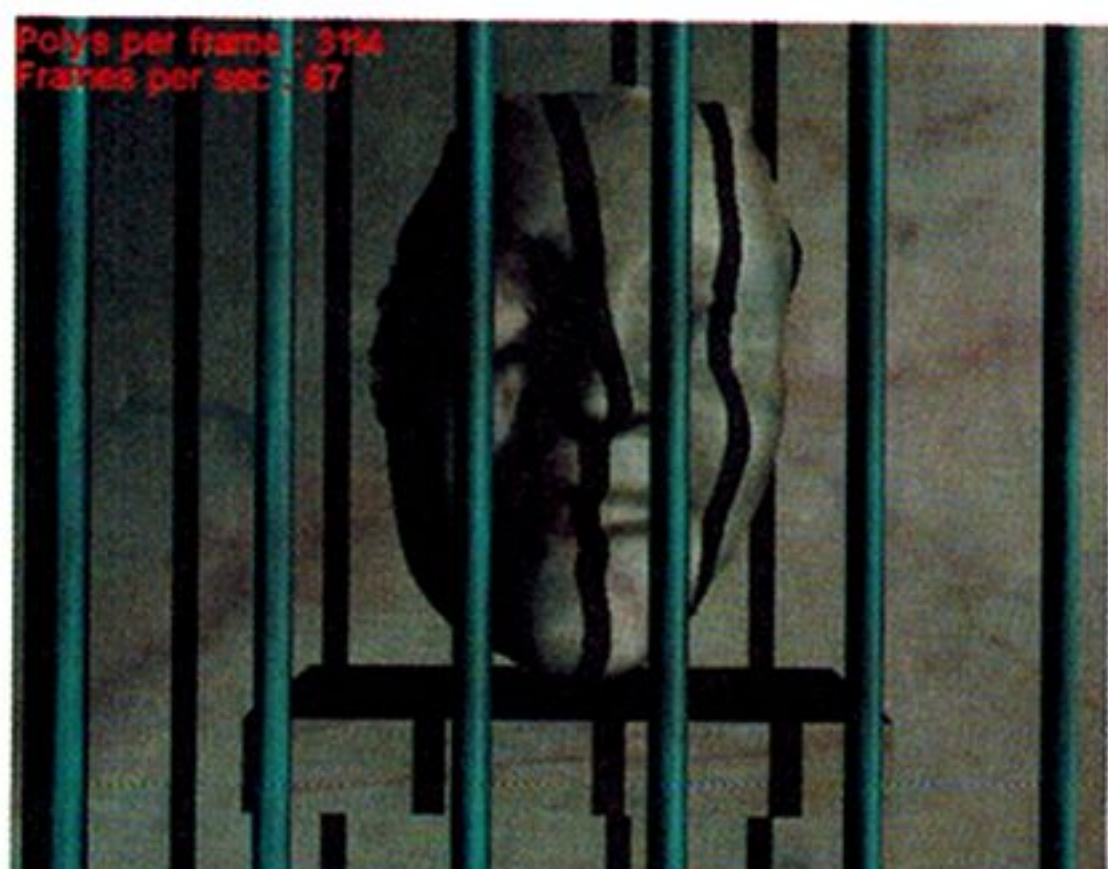
Z-Buffer　Texture　Frame Buffer
必要メモリ帯域幅　180MB/s　180MB/s　90MB/s
DEPTH → TEXTURE → SHADING → To Display

図1-B　ディスプレイリストプロセッサの動作

Z-Buffer　Texture　Frame Buffer
必要メモリ帯域幅　0MB/s　56MB/s　28MB/s
DEPTH → TEXTURE → SHADING → To Display

チップはフォンシェーディングも行わずにバンプマッピングをサポートするようになってきている。

バンプマッピングは，3D画像の表現力を大きく上げていく。非常に待ち望まれていた機能だ。ポリゴンに「質感」を与えることができる。DirectX5ではサポートされないものだが，Dreamcast版ではすでにサポートされている。これはDirectX6で加わったもので，CD-ROMに入っているDirectX6 SDKのサンプルにはバンプマッピングを使ったものも入っているので効果を確認してほしい（¥DX6SDK¥SAMPLES¥MULTIMEDIA¥D3DIM¥BIN）。

●モディファイアボリューム

モディファイアボリュームという考え方が面白い。もともとPowerVRではポリゴンで囲まれた領域を影や光の当たっている領域と見なして明度処理を行うことができた。光源から影ポリゴンを生成してやれば，物体上にリアルな影を落とせるという機能だ。PowerVR2ではこれが拡張されて，明度だけではなく，ポリゴンのさまざまなアトリビュートを変化させることができるもの＝モディファイアとして再定義された。もちろん，明度をいじれば影や光の当たったところを表現できる。これを応用すると，ポリゴンの可視不可視属性に適用して，特定の形にポリゴンをリアルタイムでくりぬいたりすることもできれば，半透明にすることもできる。こうしていろいろ用途が広がったわけだが，やはり影が落とせるというのがいちばんの魅力だろう。

DirectX6ではステンシルプレーンを使った実験で物体に影を落とすデモを実現していたが，もっとエレガントに実行できる。

こういったフィーチャーはDirect3Dでは直接サポートしていないが，D3DIMというのはもともとなにもサポートしていない低レベル関数群だけなので，独自に処理してやればいい。

グラフィック総論

ざっと機能を見てみたわけだが，加えて，テクスチャ圧縮をサポートしていることもつけ加えるべきだろうか。これはD3D標準のS3TCではなくVQ圧縮（202ページ参照）によるものだ。Direct3Dでのテクスチャ圧縮はS3TCによるものとベンダ独自の定義によるものを許容しているので特に問題はない。S3TCは8×8ドット中から代表色を2色（16ビットカラー）を取り出し，さらにそれを補間して4色のパレットを作成して2ビットのコードを割り当てるという単純な方式のものになっている。色の変化の少ない画像に対しては有効だが，実質上対処できない画像ははっきりしている。それに対し，VQ圧縮では一様に圧縮が可能だと思われるが，画質は低下する可能性が高い。どっちがよいともいえない感じだ。

ただ，Dreamcastはなぜか8MBもVRAMを搭載しており，圧縮が必要とも思えない。RGB出力があるみたいなので，高解像度の表示が可能なのか（PowerVR2は1600×1200ドットまでいける），それとも多段レンダリング（映り込みなどを表現するため，PowerVR2はユニファイドVRAMを採用している）のためなのか判然としない。多くて困ることはないのだが，VGA解像度程度なら4MBあれば十分なのだ。

いずれにせよ，現状のゲームコンソールの機能/性能と比べると圧倒的な違いがある。現状ではバイリニアフィルタすらサポートされていないのだから。

対Dreamcastの動き

もっとも気になるのはソニーの動きだろう。もちろんPlayStationの後継機は開発されている。伝え聞く話を総合すると非ポリゴン系の3D処理を開発しているらしい。非ポリゴンとなるとボリュームレンダリング系になるのだが，ボクセルレンダリングあるいはリアルタイムレイトレーシングではないかと推測する向きもある。

SCEいわく，それはCGには見えないCGを実現するものだ。限りなく実写に近い映像の生成を実現し，さらにリアルタイムにキャラクターの表情などを合成できるシステムを目指すという（なんか任天堂のProject Realityみたいだなあ……）。Dreamcastの動きを見て余裕の態勢だ。

また，伏兵もいる。VM Labsでは1999年リリースを目指して，DVDベースのコンソールを開発している。E3のデモではリアルタイムレイトレーシングによる地形表示デモが行われたという。1500MIPSという驚異的に並列化されたチップにより，これまでの常識を大きく打ち破る画像表現を達成しようとしている。

ポリゴン系を極めた感のあるPowerVR2，そしてボリューム系（？）でさらに質感表現を極めようとするPlayStation2，リアルタイムレイトレーシングを現実のものにする，その名も「ProjectX」の登場。今後も話題は尽きない。

ソニーの動きに余裕がありすぎないかという懸念もある。噂レベルでは，ナムコはPlayStation2を待つつもりはなく，鉄拳4はDreamcastで出るという予測も出ている（もちろん噂にすぎないが）。アーケードゲーム用にもDreamcast互換のNaomi基板が投入されれば，モデル3を超える（もちろんST-Vとは比較にならないくらいの）ハイレベルな映像を実現してくるだろう。優秀なハードウェアはソフトウェアがあってこそ意味があるものだが，アーケードゲームにおけるセガの実力は侮れない。

今後数カ月は各社の動きから目が離せない。

PowerVR2によるMotoRacer2の映像。この程度のクオリティなら余裕でゲームが実現されるだろう

Modifier Volume Demo

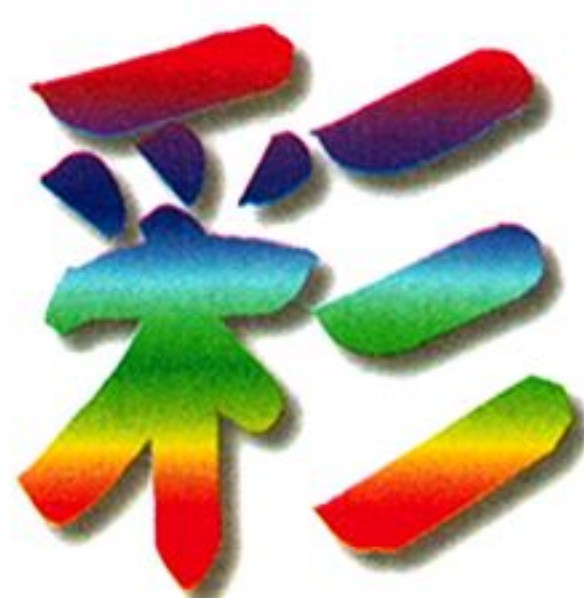
モディファイアボリュームによってレントゲンのようにポリゴンを透過させるデモ

物体の上にきちんと影が落ちる。この表現ができれば用途も広がる

環境マッピングによる映り込みの例。同じシーン内のオブジェクトが映り込んでいることに注目

画像はwww.pvr-net.comより

X68000用 フルカラーペイントツール

彩

フルカラーグラフィックツール

新生Oh!XのTHE USER'S WORKS第1弾は，小松浩司さんによるX68000用のアンチエイリアス処理のグラフィックツールです。

初めての方に解説しておきますと，このコーナーでは皆さんの作っているソフトウェア（ハードでもかまわないですが）を紹介していきます。フリーウェア/シェアウェア/同人ソフトなんでもありです。ジャンルなどもまったく問いません。掲載を希望される方は編集室までご連絡ください。

さて，今回の彩（サイと読みます）は現在制作中のものです。同人ソフト化する予定のようですが，今回の付録CD-ROMにはテスト版ということで収録されています。X68000をお持ちの方はぜひ使ってみてください。

基本仕様というか特徴は，

- ジャギレスフリーライン＆美麗エアブラシ
- 内部45ビットGRAMによる美しい描画
- 線画，背景を独立したプレーンとして管理
- レイヤーを使用可能
- アナログマスク実装
- 筆圧タブレット対応（WACOM，NScalcomp）

ということですが，このツールの最大の特徴は，フルカラーへの対応さらにアンチエイリアス処理に対応していることでしょう。アンチエイリアス関係，フルカラーへの対応，というのはOh!Xでも昔いろいろ検討した覚えがありますね。X68000は，制限された色数と解像度という十字架を背負っていますので，本格的にグラフィックをやっていくならいずれは避けられない道なのかもしれません。

とにかくこの手のものは処理のバランスが重要です。普通に考えると，グラフィックツールを作るにしてもフルカラー処理のほうが実は簡単です。内部処理自体はよいとしても，最終的な表示は16ビットカラーですし，エディットもその状態で行われますから，そのあたりのつじつまをあわせつつ，できるだけ高品質で出力するなりしなければなりません。それは多くの場合レスポンスを犠牲にするものです。レスポンスを重視すれば今度は見た目がエディットしているものとかけ離れてしまいます。どうしようもないといえば，どうしようもないんですけどね。さらにメモリの問題もあります。一般的な1ドット32ビットというフォーマットを採用すると，メモリ使用量は倍になります。それだけでも大変なのに，このツールではなんと45ビットというフォーマットを採用しています。一気に3倍の大きさになります。うーむ。さらにこのソフトではレイヤーに対応しています（これもメモリを食います）。さすがに背景と線画だけのようですが（本体を除いては），割り切った部分とこだわった部分が見え隠れしていて面白い仕様です。

仮想メモリ処理，内部バッファ圧縮処理など，技術的に可能である対処法はありますが，実際に使うとなるとパフォーマンスも重要になりますので，あまり現実的ではありません。結局はどこまで高速化できるかというのがあって，処理の重さとバランスをとっていくことになります。

フルカラーはもちろん，アンチエイリアスにしても，理論的な背景はだいたいクリアになっていますので，実装技術の巧拙によって実現するか否かが変わってくることになります。CD-ROMに収録されているサンプル画像を見ればおわかりでしょうが，このツールは結構いい線を出しているのではないかと思います。

フルカラーによる完成画像例

そして，ブラシが綺麗なのも注目でしょう。いわゆる「Photoshop塗り」ができるという感じで，柔らかな質感を実現しています。最終的にどこまで作り込んでいくのかに注目したいと思います。

X68000では別個に，制限されたメモリとCPU能力という問題がありまして，いろいろなことをやろうとするとそれが障害になってきます。ただ，メモリとCPU能力については改善することが可能ですので，色数などに比べて割り切りがしやすいかもしれません。

以前作ったEX-Systemでは可能だとはわかっていても　結局，メモリ容量の問題で断念せざるをえない部分がありました。1MBユーザーでも使えるようにとか考えるとこういうアプローチはしにくいものです。

さて，ひとついわせていただくと，より画質を追求するのであれば，アンチエイリアスではなくて実データは高解像度（オーバーサンプリング状態）で保持しておいて，表示時に処理するというアプローチのほうがさらに綺麗なのですが，それはさすがにメモリ容量が許さないというところでしょうか（X68000は標準2MB，最大で12MB）。

なお，試用してのご意見などありましたら編集室までお寄せください。また，このコーナーへの皆様の投稿をお待ちしています。

彩の基本画面

マスクを指定しているところ

収録には間に合わなかった新バージョンの画面レイアウト

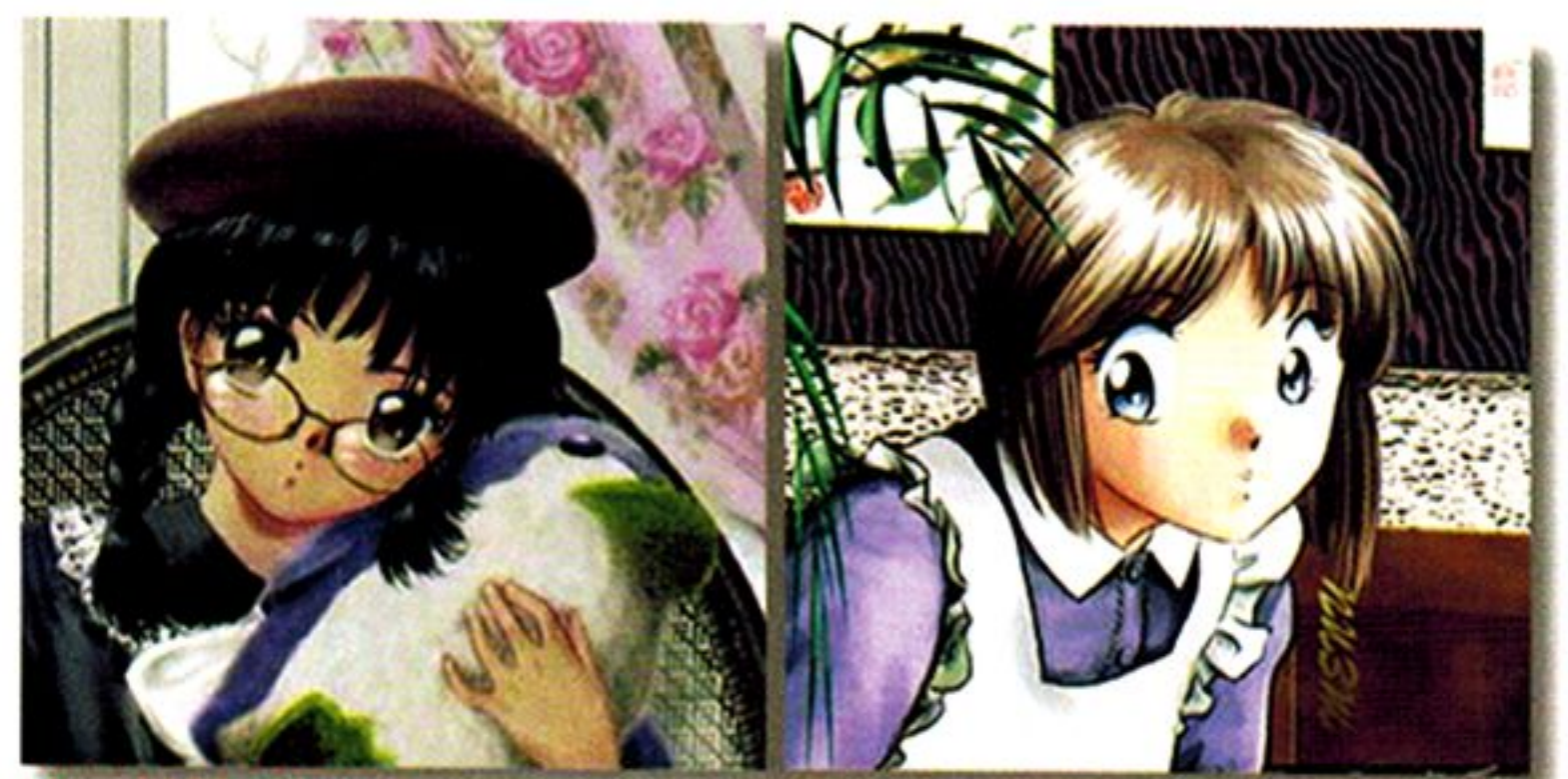
パソコンの進化は非常に速い

CPUの速度，メモリ容量，ディスク容量これらが劇的に拡大されつつある。そして画面出力関係の進化がめざましい。PCとDOS時代のコンソールにおける文字中心の文化のときには，カラーグラフィックなどはワークステーションの分担領域だったといってもいい。やがて，Macintosh IIなどでフルカラーのパソコン環境が登場したものの，VRAM構造や処理速度からいって，実用性が出てきたのはかなり時間を経てからのことになる。いずれにせよ，プロフェッショナル領域に近い部分だった。
Windowsの登場以来，256色，高解像度化，そしてハイカラー，フルカラーへと長足の進歩を遂げている。フルカラー環境はまだ普及しているわけではないが，それがすぐに実現できる状態でセットアップされているのは確かだ。
高解像度フルカラー。その大半はウィンドウ枠やインターネットのJPEG画像を綺麗に表示するために使われている。せいぜいがゲーム画面のためだ。
高速なCPU，大容量メモリ，大容量ディスク……。高度なグラフィック処理を行うにあたって障害となりそうなものはほぼなくなった。以前は望んでも簡単には得られるようなものではなかったものが，いまは無造作に与えられている。我々の前には実にたくさんの可能性が転がっているのだ。まず，それを自覚すべきだろう。
しかしこのグラフィック機能を活用するためのノウハウを我々は持っていない。フルカラー，すなわち，写真とほぼ同じクオリティで生成可能な画像環境。
それは表現力が限りなく拡大されていることを意味する。どんな手法をも受け止める媒体であるからこそ，どのようなアプローチで向かっていくかが問われる時代になってきているといえるだろう。
ここではグラフィック画像の処理についてさまざまな実験を行っていきたい。
グラフィックの用途のひとつは「美しい映像」を作り出すことだ。それを2D，3D，4Dへと展開していく。
絵の描き方などに正解はありえないし，上手下手というのも二の次の問題だ。
まずは，なにかを作り出していく姿勢がなければ上達はありえない。そして，ノウハウは共有されるべきものだと信じている。そのうえで「個性」を磨く。そういう次元で語りたい人のための場所を用意したい。

Visual Laboratory

Kazuhiko Tsuzuki

You Kawahara

Haiboku

Hisashi Morikawa

Emi Nozawa

Kei Yoshimizu

都築和彦

K a z u h i k o　T s u z u k i

日差しの中で

使用環境
本体：IBM Aptiva Pentium/200MHz
メモリ：256MB
HD：4GB
WACOM ArtPad
Windows 95

今回は，光と水をテーマにしてみました（実は女の子がメインかもしれませんが）。

では，簡単ですが，描き方とかを紹介いたします。私は，自作ツール“きらきら筆（Light EDitor）”を使ってますが，皆さんの使っているツールで機能が似てるものがあれば，置き換えて読んでみてください。なお，Light EDitorはCD-ROMに収録されていますが，ワコム製のタブレットをお使いの方だけLED0.EXE，そうでない方はLED0B.EXEを実行してください。間違えるとシステムエラーが発生します。

まず，線画を用意します。スキャナで取り込んだり，マウスタブレットで直接描くなりします。私の場合，イメージを決める段階で，紙に落書きしていて図柄が決まったときは，紙で清書して，スキャナで取り込みます。モニタを見ながら，マウスタブレットで落書きしながら，図柄が決まったときは，そのまま，パソコン上で描き込んでいくことが多いです。

今回は，直接タブレットを使って描きました。

ラフに線を描いて，修正していきます。筆は，「描き込んでいくと，黒くなっていく筆」を利用すると，鉛筆描きの感覚に近くなるのでいいでしょう。

線の修正で，2時間くらいかかりました。こんなことなら，紙に描き写してスキャナを使ったほうが早かったかもしれません。ただ，スキャナを使った場合でも，線の修正は必ずしています。

空の色味を決めます。グラデーションを微妙にかけるといいでしょう。雲のアタリを，適当に決めておきます。

●Light EDitor の場合

・メインコントロール

上方のボタンで，「DRAFT」モードにします。カレントカラー，濃度，筆の種類（ノーマル筆），を選択。線画は，メインコントロールの「Pen」を使用。

円形のグラデーションをかけるときは，筆の大きさを最大にします。筆の大きさで，効果が変わります。

・ドラフターパネル

BOXFILL，円フレア，補正フレアを選択。

・コレクションパネル

フレア色補正で，色をセットし，スムージングして，グラデーションデータを設定。

全体を青で塗りつぶしたあと，上の設定でドラフターパネルから機能を実行すると，円形のグラデーションがかかったデータができます。

空に調子をつけます

青空と雲に，絵画風に筆のタッチをつけてみます。このような場合，フィルタを全画面に使うと平面的になってしまいます。空の奥行き感を出すために，面倒くさがらずに筆でタッチをつけていきます。手前（上）の雲は，大きい筆のタッチで，遠く（下）の雲は，小さい筆で，描き込んでいきます。このときの筆は，1点描くごとに，選択されたカレントカラーの色味を微妙に変えるようなものを使うと便利です。白い雲を描くのに，白一色ではなく，淡い水色，淡い桃色，淡い黄色と，タッチごとに少し色味が変わるようにしますと，絵に厚みが生まれます。

●Light EDitor の場合

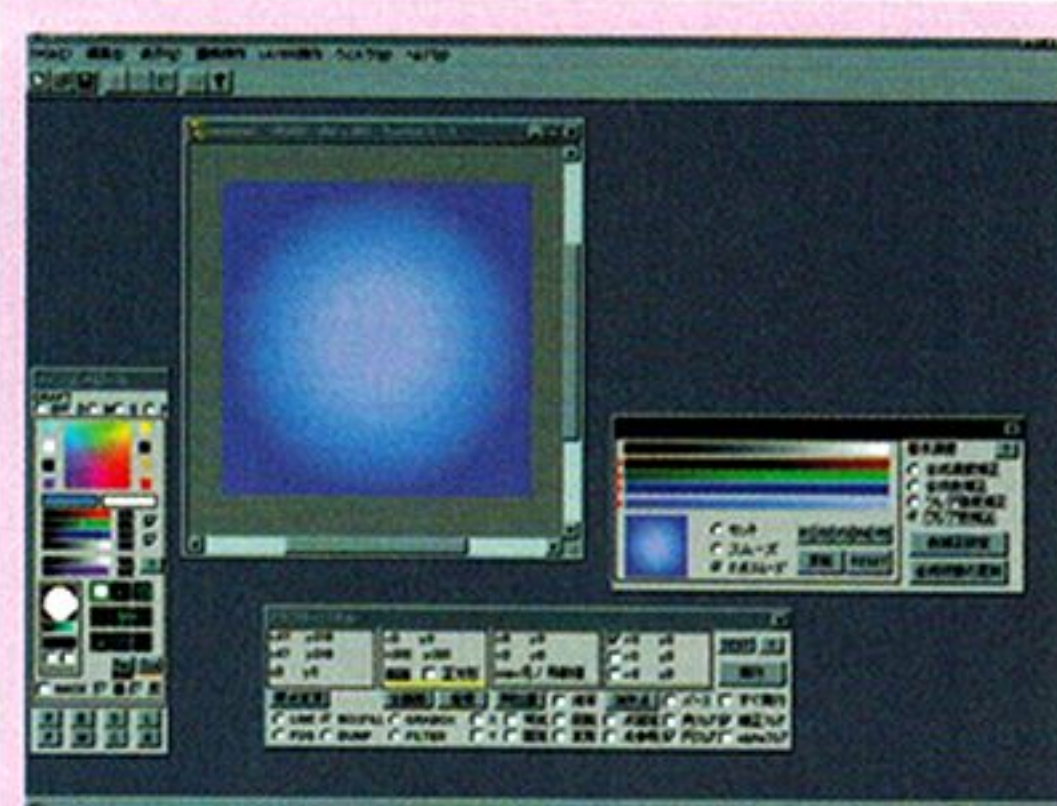
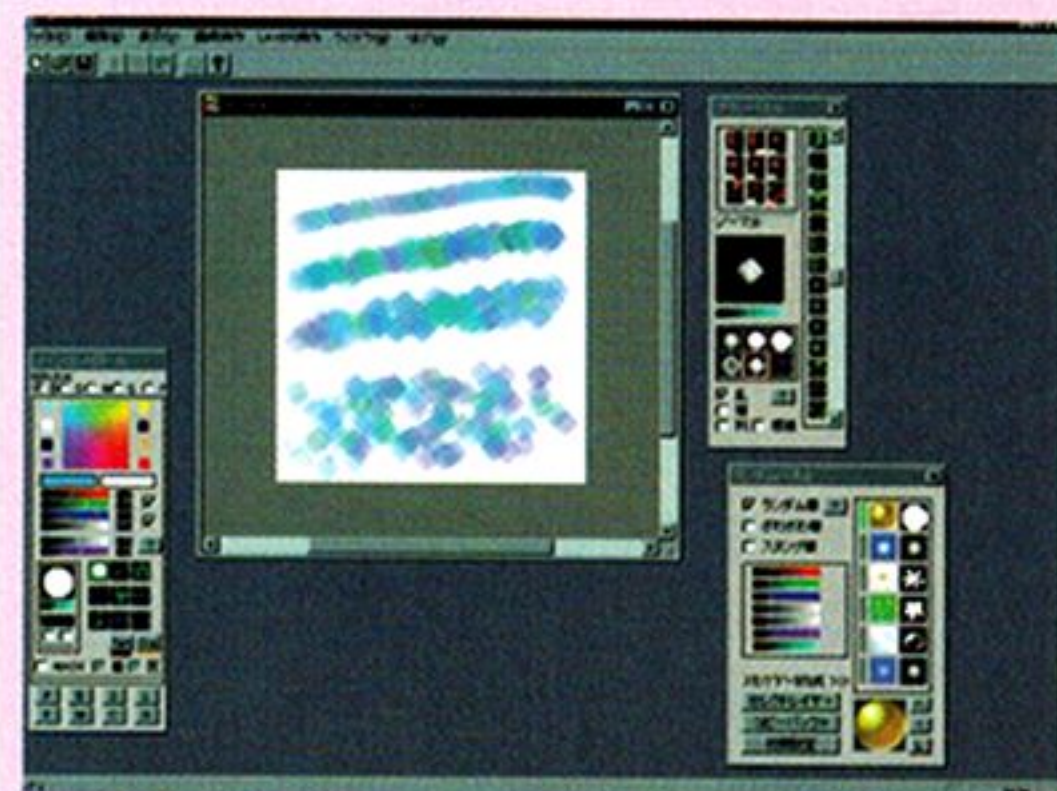
・メインコントロール

上方のボタンで，「BRUSH」モードにします。カレントカラー，濃度，筆の種類，筆の大きさを選択。

・ブラシパネル

筆先を設定。

・ランダムパネル

「ランダム筆」を選択。

DRAFTモードで最大の大きさのPenを使い，ドラフターの機能を組み合わせてグラデを作っていく

雲用にランダム筆で変化のあるPenを作る

筆に，特別な効果をプラスします。筆の種類で効果が変わります。描画中，ランダムに，各パラメータを変化させます。

色味，筆の散らばり度，筆の大きさ度の変化幅を選択。パラメータが小さいほど，変化幅が小さくなります。

パラメータは，上から，

色味（赤，緑，青）

スタンプのアルファデータ濃度

描画濃度

筆の散らばり度

筆の大きさ度

と，なっています。

今回の空では，筆の散らばり度，筆の大きさ度を最小にしています。すると，色味だけが変わります。散らばり度を上げると，図の例の下側のように描画が散らばります。

海に調子をつけます

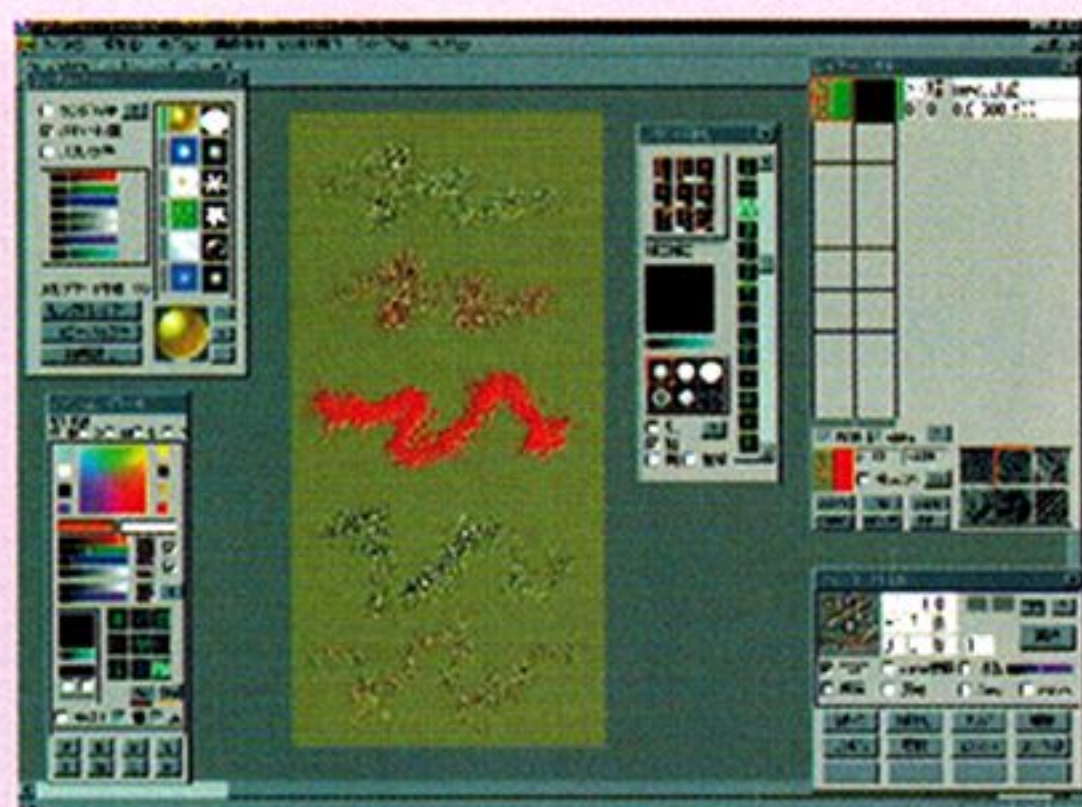
簡単に，海の色や，背景の下塗りをします。白波をあとから描き加えます。透明感を意識して描きましょう。

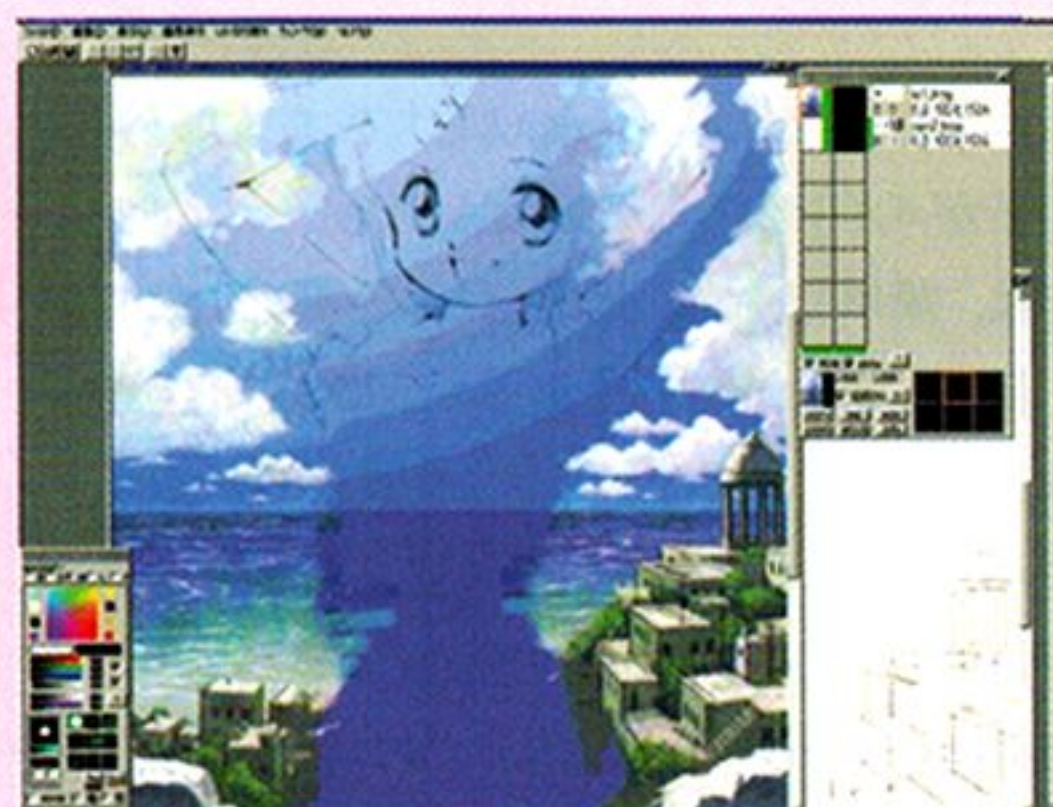
手前の壁に調子をつけます。

朽ちた感じを出すために，軽くレリーフなどのフィルタを部分的にかけています。これも，あまりやりすぎるとうるさくなるので気をつけましょう。

タブレットを使ってディスプレイ上に下書きの線画を描いていく

背景。グラデで空を描き，雲のアタリを入れる

雲にタッチをつけていく

ざわざわ筆を使うとこんな感じ。ぼこぼこ筆と組み合わせると下のような感じ

パース取りのための専用ツールを使ってみる

下書きレイヤーにパースのアタリを入れて家並みを描き込んでいく

●Light EDitor の場合

・メインコントロール

上方のボタンで,「BRUSH」モードにします。濃度,筆の大きさ,ラインモードを選択。

筆の大きさは,やや,小さくしておきます。筆の大きさを決めるところの下に,ラインの粗さを設定するところがあります。マウスクリックして,線の粗いモードにします。

・ブラシパネル

「ぼこぼこ筆」を選択。

ぼこぼこ筆は,フィルタの効果を筆で描画できるようにしたものです。ぐりぐり描くと,ぼこぼこになります。ぼこぼこ筆のみで使うときは,筆を大きくしてもいいでしょう。重いですが。

・レイヤーパネル

紙質を選択します。ぼこぼこ筆は,紙質のパターンで,絵をぼこぼこにします。

・フィルターパネル

ぼこぼこにするには,レリーフボタンを押して,パラメータをセットします。紫のバーは,フィルター強度です。通常,レベルを中央にしておきますが,強くかかりすぎるときは,弱くします。

「地色」をチェックします。下地の色味で,ぼこぼこにします。チェックをはずすと,カレントカラーの色味でぼこぼこにします。

・ランダムパネル

「ざわざわ筆」を選択。

筆の散らばり度,筆の大きさ度の変化幅を選択。散らばり度を上げると,ヒビのように散らばります。ぼこぼこ筆と組み合わせると,壁にヒビが入ったような感じになります。

家並みを描きます

家を描くとき,パースがあまりにも適当だとアレなので,それなりに修正しておきましょう。

●Light EDitor の場合

Light EDitor には,パースをとるための機能があります。

・メインコントロール

上方のボタンで,「MAGNIFY」モードにします。

画面上で右クリックして,縮小させます。

・メインコントロール

上方のボタンで,「DRAFT」モードにします。

・ドラフター

「パース」をチェックします。ほかはチェックしないでください。すると,画面に点線の箱が表示されると思います。マウス右クリックで,箱の頂点が解除されていきます。マウス左クリックで,あらたに指定できます。緑の線は水平線です。この箱を指定することで,パースをとるための消失点を算出します。このイラストの消失点は,

右:3811,849

左:547,849

上:868,-11732

です。

消失点ボタンを押すと詳細設定ができます。ここで数値入力してもいいでしょう。数値は,メモしておくことをおすすめします。

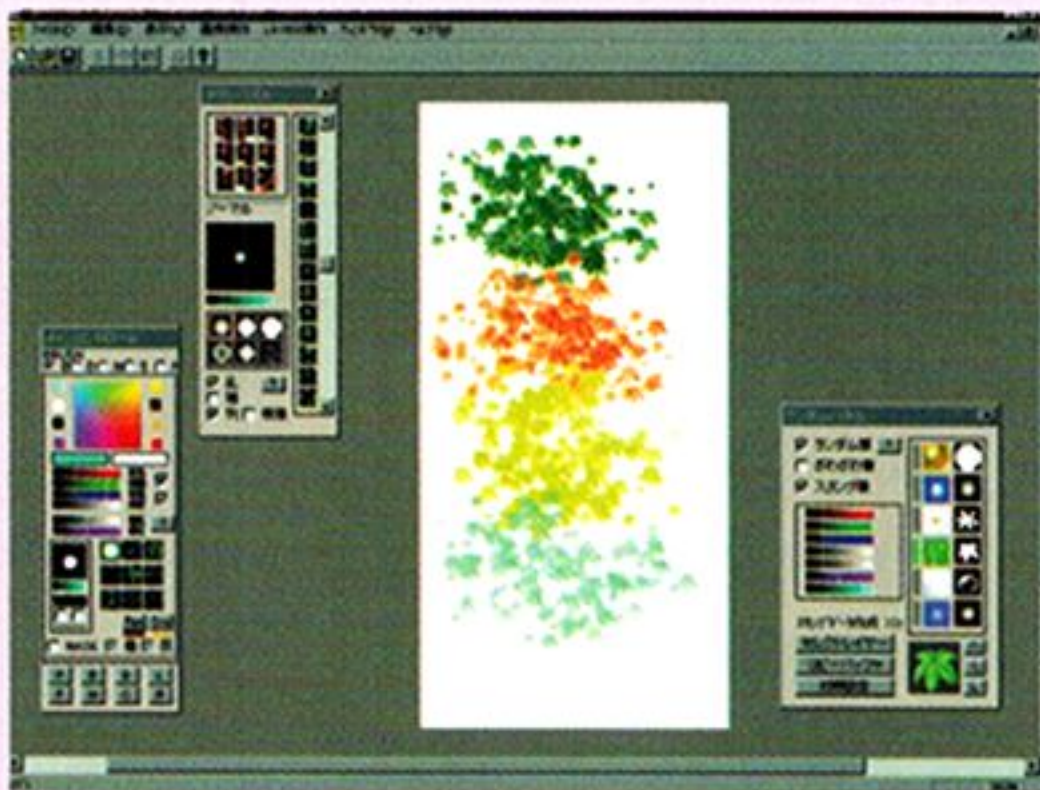
また,このイラストのように,水平線が水平な場合,「水平線を水平に固定」をチェックすると,箱でパース指定するとき,水平線が水平になるように設定することができます。

この消失点を利用して,線分を引くとき,正しいパースが得られるわけです。下書き,アタリをとるときに便利です。ただし,このパースの場合,画面の左半分ではパースがかかりすぎてしまいますので,機能を使わないで適当にごまかすようにします。

家並みの完成

これが下書きのプレーン。左側ではほぼ使われていない

スタンプ機能で茂みを描き込んでいく

コントラストを補正して仕上げる

ランダム筆，スタンプ筆の複合で葉っぱのパターンを描き込んでみる

塗り込まれた肌のレイヤー

服や髪の毛を加えて人物を仕上げていく

線画部分も修正しながら塗り込んでいく

・メインコントロール

濃度，筆の大きさを選択。筆の大きさは，最小にします。

・ドラフター

「パース」「点参照」をチェック。その上の3点の消失点数値の横にチェックボタンがあるので，目的の消失点を選びます。

「LINE」モードにします。これで，常に消失点に向かうラインを指定できます。「実行」ボタンで実行です。「すぐ実行」をチェックしておくと，マウスボタンを離したときにすぐ実行され，作業を手早く行えます。

合成させながらアタリをとり修正しました。

茂みを描きます

茂みを描いてみました。こういうところは，スタンプを使ってぺたぺたすると便利でしょう。葉っぱに限らず，自然物をスタンプで表現するときは，大きさや色味を少しずつ変えて表現すると，それらしく見えます。

コントラストを調整しました。これで，背景の完成です。

●Light EDitor の場合

・ランダムパネル

「ランダム筆」，「スタンプ筆」をチェックします。

「葉っぱ」のスタンプを選びます。

紫のバーの値を大きくすると，葉っぱが散らかります。水色のバーの値を大きくすると，葉っぱの大きさがランダムに変化します。

スタンプデータは，自分で作ることができます。100×100，アルファデータ付き（4チャンネル），RAWセーブします。6つまでしか登録できないので，STAMP＊.RAWを上書きするか，読み直して，セレクトレイヤーからコピーしてください。

・メインコントロール

色を設定します。

カレントカラーによって，スタンプの色が変化します。スタンプデータの色のままにするには，カレントカラーをグレー(128，128，128)にします。

色の変化幅は，ランダムパネルで調整できます。最小にすると，色が変化しません。

筆の大きさを設定します。

ラインモードを粗くします。

・ブラシパネル

筆先の設定を，エアブラシっぽいものにします。スタンプデータによって，筆先の種類で効果が変わります。なお，コントラストの調整は，「メニュー→画像操作→色補正変換」です。

人物を描きます

レイヤーを使って，線画と，肌用画像を合成させて，肌色を塗っていきます。

大まかに色を塗り，ひきずり筆で，なじませていきます。この段階での輪郭線からのはみ出しは気にすることはありません。最後に，背景との合成のときに輪郭線を修正することにしますので。

髪の毛，服を描きます。これで，人物は完成です。

色を塗りながら，線画も少しずつ修正します。

●Light EDitor の場合

Light EDitorで作業すると，このような感じになります。

・レイヤーパネル

ファイル名のあたりをクリックすると，合成モードの設定画面が出ますので，確認してみてください。

この例ではhada1.bmpが1枚目，sen1.bmpが2枚目で，sen1.bmpの合成モードは「線描き用」にします。描画レイヤー「＜描」をhada1.bmpにあわせます。下のほうの，「RGB」「alpha」とあるスイッチは，「RGB」を選択します。「RGB」だとグラフィックのみに描画することができます。

・ブラシパネル

ノーマル筆で，べたべた描き，ひきずり筆で，なじませます。

肌色を描き終わり，このまま服を描くときは，描画レイヤー「＜描」をhada1.bmpにあわせたまま，「メニュー→LAYER操作→FIX」します。

UNDOバッファに，hada1.bmpが退避されました。

ここから，服の色を塗っていきます。肌色部分にはみ出したら，「消しゴム」筆を使います。

Light EDitorの消しゴムは，FIXした時点のUNDOバッファから描き戻します。肌色が描き戻されたと思います。私は，こんな感じで，FIX→描画→はみ出したら消しゴム→描画→FIXと，繰り返して作業しています。

ちなみに，マウス右ダブルクリックで消しゴムになります。マウス右クリックでは，スポイトです。

・メインコントロール

消しゴムのときは，紫のバーの値は，最大にしておくといいでしょう。よく消えますので。

筆の大きさを調整する個所の下に，ラインモードを決めるボタンがあります。クリックすると，直線，破線，曲線と変わります。その下のほうに，チェックボタンが2つあります。左の

下描きレイヤーを使って，実際にはこういう感じで塗り絵をしていく

空の部分に太陽を加えてみる

光源を意識した人物と合成して基本部分のできあがり

チェックボタンは，簡易ラインモード，右のチェックボタンは，ラインに筆圧的な濃度調整をするモードです。

両方ONにして，ラインモードを「曲線」にすると，簡易的に曲線が引けるモードになります。ラインモードが「直線」だと，直線が引けます。点を指定していき，「終点でマウスボタンを押しっぱなしにする」と，実行されます。髪の毛の線のあたりは，このモードで描きました。

注：レイヤー合成中の消しゴムは，1枚目レイヤーのみに使用してください。

背景と人物を合成します。レイヤー合成機能で行うと簡単です。

背景に太陽光を足します

人物に少し手を加えました。

合成後，完成させます

●Light EDitor の場合

Light EDitorで作業すると，このような感じになります。

・レイヤーパネル

ファイル名のあたりをクリックすると，合成モードの設定画面が出ますので，確認してみてください。

完成背景が1枚目。完成人物が2枚目，hito1.bmpをロードすると，hito1.alpがロードされていることがわかります。ロードされないときは，「ALPロード」で読み込みます。hito1.bmpの合成モードは，「アルファ合成」にします。修正版線画が3枚目で，sen3.bmpの合成モードは「線描き用」にします。

描画レイヤー「＜描」をhito1.bmpにあわせます。

下のほうの，「RGB」「alpha」とあるスイッチは，「alpha」を選択します。「alpha」だとアルファチャンネルのみに描画することができます。

「RGB」「alpha」両方をONにして描画すると，描画レイヤーに色と，アルファチャンネルが同時に描画されます。が，すでに人物は完成しているので，背景と合成させるには，アルファチャンネルのみの描画にするわけです。

・メインコントロール

色を決める数値バーの下にグレーのバーがあります（4番目のバー）。

アルファチャンネルに描画する濃度を決めます。グレーの色だと思ってください。

アルファ合成で，合成させたいときは，ここの値を高く（白く）して描画します。人物を合成させたいので，この値は255とします。

もし，幽霊など，半透明にしたい場合は，数値を下げて描画します。アルファチャンネルのみの描画にすると，半透明合成も，筆で塗りながら出すことができます。

・ブラシパネル

ノーマル筆で，べたべた描き，ひきずり筆で，なじませます。アルファチャンネルの白い部分と黒い部分をなじませると，半透明っぽくなります。

人物の輪郭に沿ってアルファチャンネルに描くと，人物が浮き出てきます。

もし，輪郭からはみ出したら，背景のところでマウスを右クリックしてください。アルファチャンネルの値の低い（黒い）部分の数値が拾えます（0のはず）。このままはみ出したところをなぞるように描くと，消すことができます。

また，人物の中でマウスを右クリックすると，アルファチャンネルの値の高い（白い）部分の数値が拾えます（255）。

私は，2枚の絵を合成するときは，2枚の絵を別々に完成させたあと，片方の絵を，あぶり出すようにして合成させています。

一度に，背景と人物をレイヤー合成させながら描画する方法もありますが，輪郭線のあたりで，はみ出さないように色を塗るのが手間だったりしますので，このような手法で作業しています。

背景に太陽光を描き加えます。

・レイヤーパネル

「RGB」オン「alpha」オフ，描画レイヤーを背景（1枚目）にします。

人物にはマスク（α）パターンが作成してある

αプレーンに書き込むと背景と合成されていく。αチャンネルの動作を理解しておこう

円グラデの応用で太陽光の基本パターンを作る

ホースを持たせてみたところ

αプレーンの部分で水飛沫を作成

バンプマッピングで水の屈折を表現してみた

・コレクションパネル
青空の円形のグラデーションをかけたやり方で，円形の光を描きます。「P2」や「P4」ボタンを押すと，光っぽい色が設定されます。色味は，好みで設定できます。
・ブラシパネル
「あかるく」筆を選択します。これで，実行すると，明るい光のような効果になります。
「くらく」筆，「ノーマル」筆などで，効果が変わります。

合成を完成させます。
「メニュー→LAYER操作→COMPLETE」を実行します。

ホースを描き加えます

●Light EDitor の場合
合成を完成させます。
「メニュー→LAYER操作→COMPLETE」を実行します。ホースは，簡易曲線描画モードで，スタンプ筆と組み合わせて描きました。玉状のスタンプで，カレントカラーを青にすると，このような色味でホースが描けます。
そのあと，ひきずり筆で，なじませています。

ホースから飛び散る水飛沫のデータを作ります。このデータをもとに，合成済み画像に，バンプマッピングをかけると，図のようになります。

●Light EDitor の場合
レイヤー合成を利用して，水飛沫のアルファデータを作ります。
mizu1.bmpを「ALPセーブ」して，アルファデータをセーブします。前の処理で合成済みの画像を読み込み，「ALPロード」で，飛沫を読みます。fin2.bmpのアルファデータとして，水飛沫のデータがあることを確認して，次の操作をします。

・ドラフターパネル
「BUMP」を選択して，実行します。
・メインコントロール
筆の大きさで，効果が変わります。

バンプ付きの画像に，飛沫を合成します。
合成するときに，水飛沫の輪郭線が強調されるように補正をかけると，より水っぽく見えます。

合成結果です。きらきらさせると綺麗でしょう。

虹を描き込んで完成です。

●Light EDitor の場合
水飛沫の合成。
・コレクションパネル
「合成濃度補正」をチェックします。
左側の，グレーのグラデーションバーの上から2つ目に注目してください。暗いほうに，明るい山があると思います。このように合成用アルファ値を補正すると，輪郭線を強調するような効果になります。
・レイヤーパネル
mizu1.bmpのファイル名のあたりをクリックすると，モード変更パネルが開きます。ここで，「合成色補正を使う」をチェックします。すると，図のような合成になります。

きらきらさせる。
・ブラシパネル
「きらきら」筆を選択して，きらきらさせます。

虹を描く。
・コレクションパネル
「フレア色補正」をチェックし，「P5」ボタンを押します。虹のデータがセットされます。
データは，自分でエディットすることもできます。

飛沫部分のαデータを作成していく

バンプマッピングの実行。絵に独特の質感を与えることができる

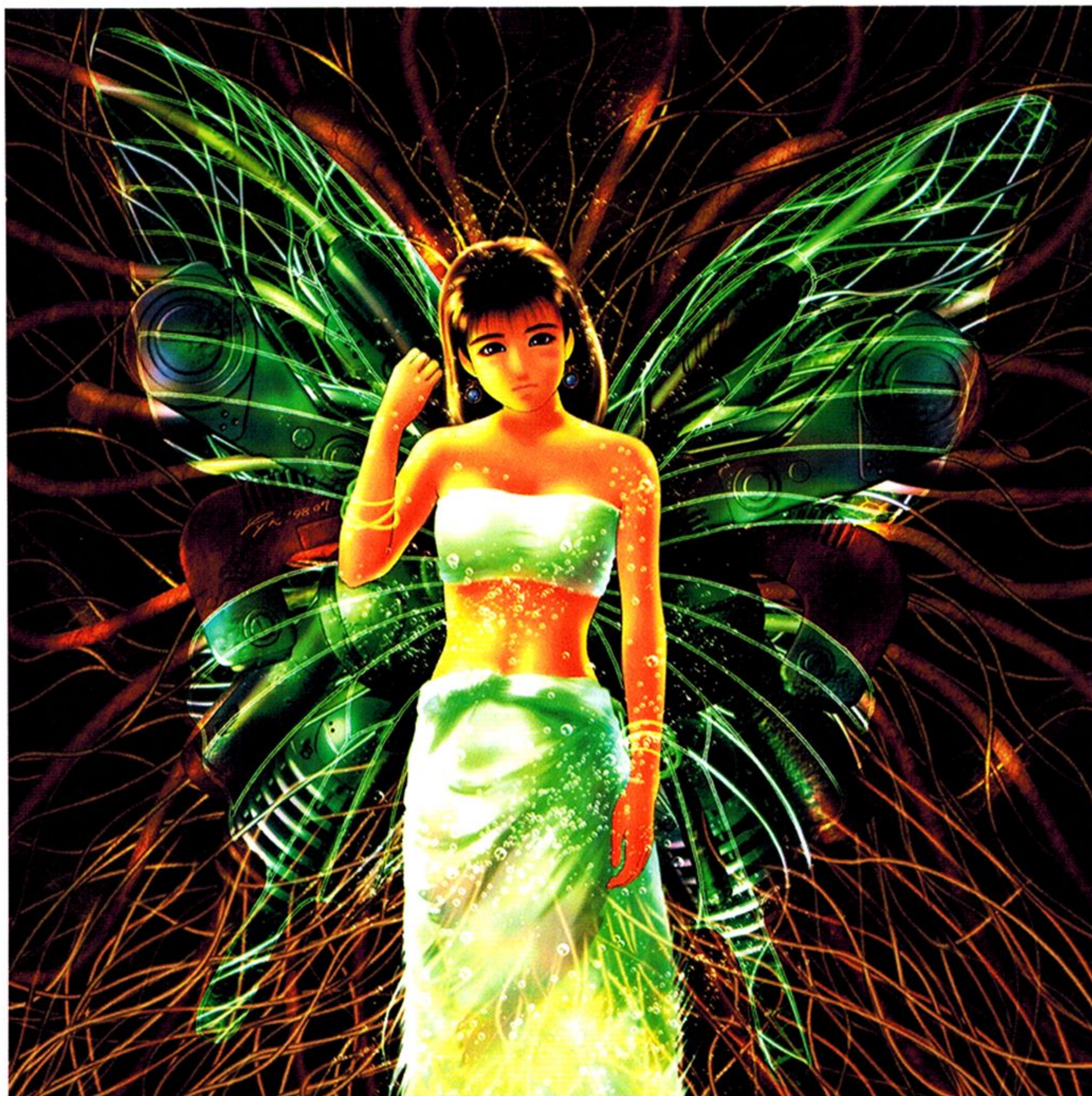
α合成を行う。このときに合成補正をかけるように指定する。コレクションパネルに注目

フレア色補正を使って虹を描くことができる。これを加えるとついに完成だ

・ドラフターパネル
「BOXFILL」「円フレア」「補正フレア」をチェックします。範囲指定して，実行します。ここで「正方形」をチェックすると，範囲指定が正方形に固定されます。

・メインコントロール
筆の大きさで，虹の太さが変わります。

川原由唯

Y o u i K a w a h a r a

だからあまえは許されません！

Power MacとPhotoshopを購入してから，イラストCGの制作における解像度や色数に対する不満はまったくなくなりました。こんな潤沢な環境が趣味の領域で使えるとは，8色デジタルの時代から考えたら夢のようです。静止画の生成環境としては，処理速度を除けばもはや飽和点を超えたと思っているのですがいかがでしょうか（動画や3Dをやるにはまだ全然ダメというのが持論）。

僕は絵に関しては素人ですし，これといったテクニックを持っているわけではないのですけれど，これからCGを始めてみたいっていう方には多少は参考になることもあるかもしれませんので，以下で手順のようなものを書いてみることにします。偉そうなこと書いているかもしれませんが，僕自身もいろいろと試行錯誤をしている段階です。「ここはこうすればもっとうまくできるよ」ってなご意見ご感想がありましたらぜひ教えてください。よろしくお願いします。

使用環境
本体：Power Macintosh 7500/100
アクセラレータ：MACh SPEED 604e 233MHz
搭載メモリ：352MB
VRAM：4MBに増設
ハードディスク：内蔵1GB＋4GB
ディスプレイ：17インチのトリニトロン
タブレット
フラットベッドスキャナ：JX-250
その他，230MBのMO，フィルムスキャナなど

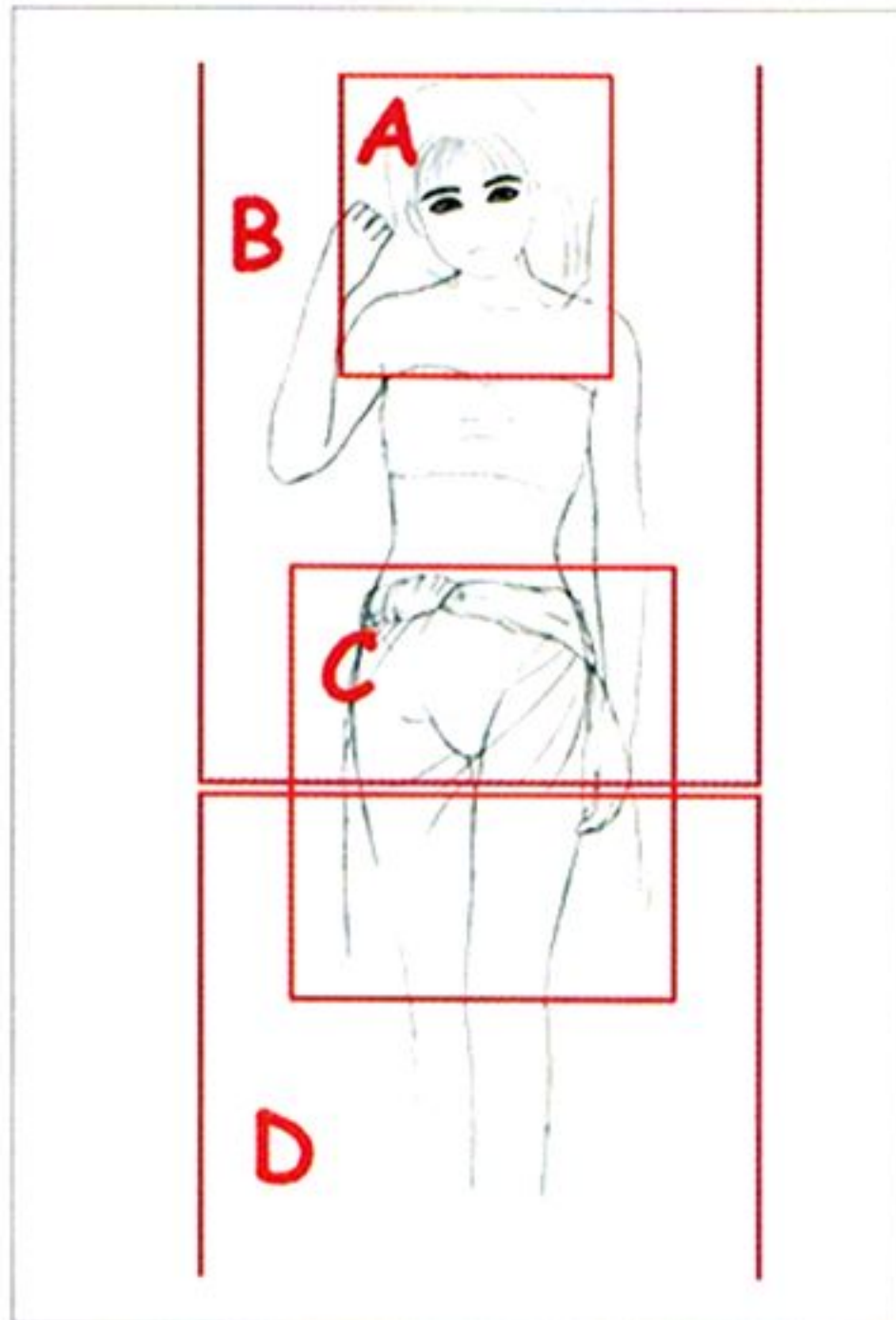
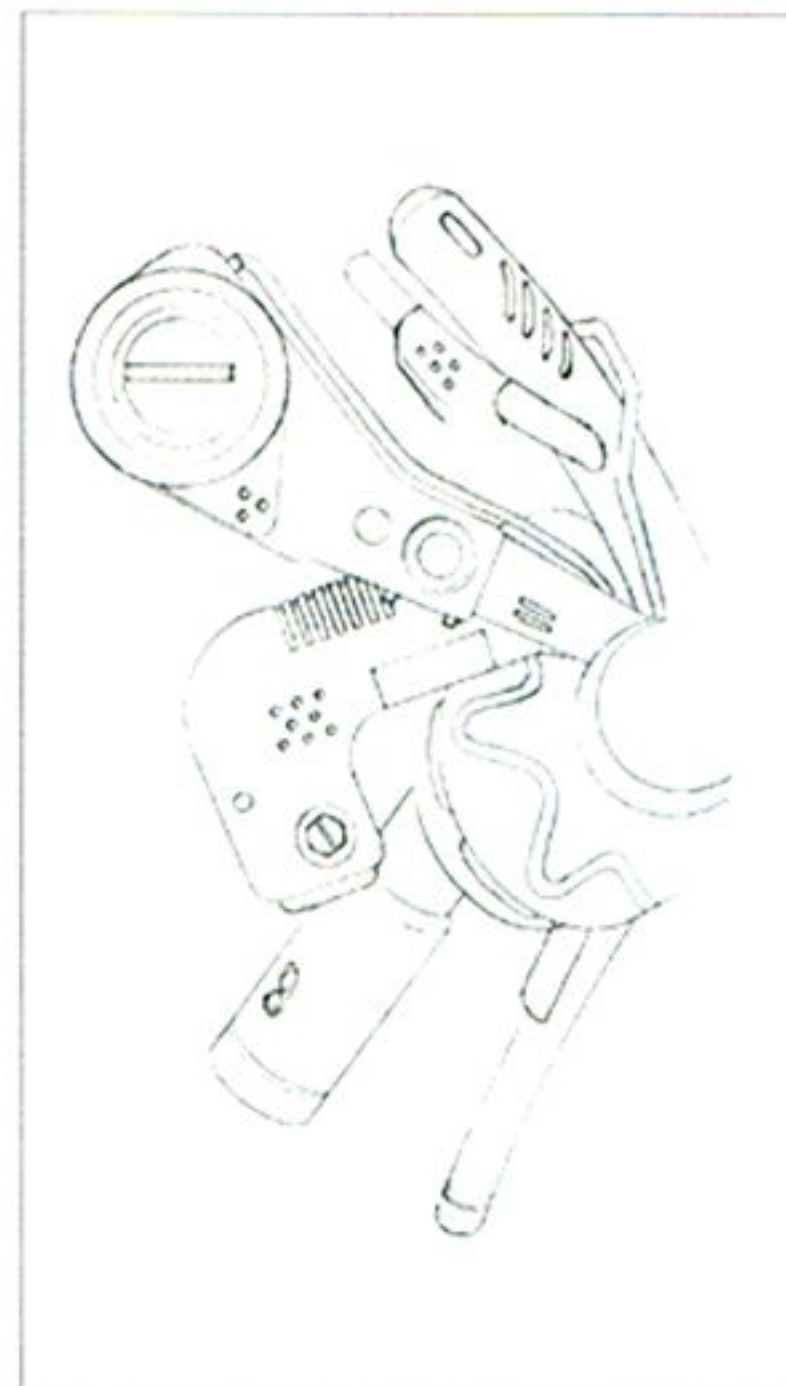
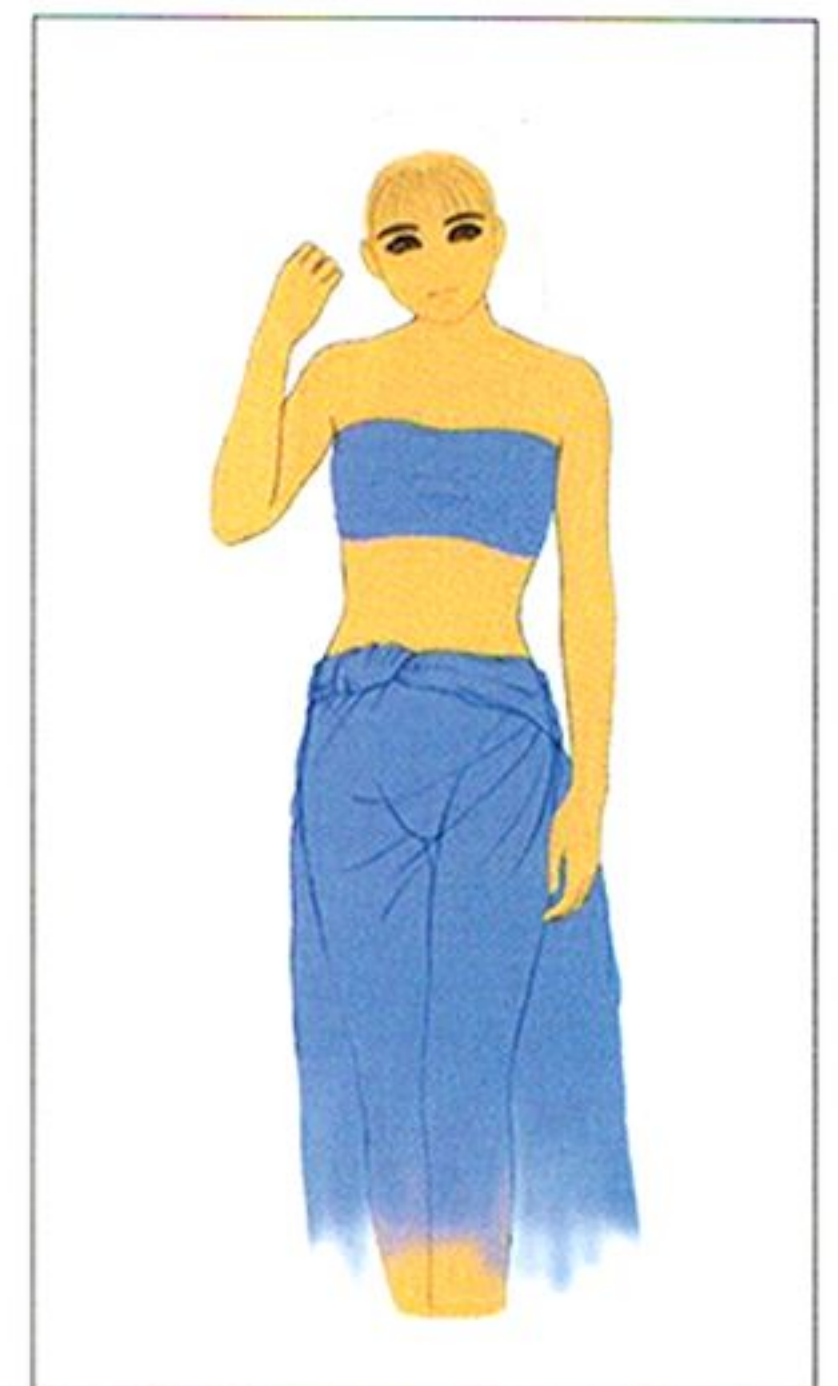
使用環境は表のとおり。2年くらいかけてかなり贅沢な環境を整えました（と，自分では思っている）。欲をいえば，21インチのディスプレイにしたいとか，メモリはもっと積みたいとかいろいろあるんですが，まぁとりあえずは描きたいものは十分描ける環境になっているといっていいでしょう。職業でCGを描いている人とそれほど差はないかもしれません。最近はハイアマチュアのほうがいい環境持ってることが多いんですよね〜。作品の出来を環境のせいにして逃げることはもはやできません。

ソフトウェア

ほとんど100％近くをPhotoshopで描いてます。Painterも持っているのですが（というより，Painterが使いたくてMacintoshを買ったのですが），無闇に増築を重ねたようなユーザーインタフェイスにどうも慣れないのと，タッチがイマイチ自分の手癖にあわないので最近ではほとんど使っていません。市販のプラグインなどはいまのところ持っていません。

画像の大きさは，2000×2000pixelくらいで仕上げるケースが多いです。depthはもちろん24ビット（俗にいうフルカラー）です。

以下で書いている実録のとおり，僕はレイヤーを多用する傾向があるので，メモリは350MB以上あってもまだ足りないです（困ったもんだ）。

蝶の羽のイメージ。昆虫図鑑を参照して，サインペンで紙に描いたものを取り込んでいます。反転してコピーするつもりなので，片側だけ描いてます。コントラストを高くし，明るさを上げるとスキャン取り込み時のゴミが減ります。そのあと，輪郭線抽出のフィルタを用いて２本線に加工

今回の女の子のキャラクターは下描きがでっかくなっちゃったので分割して取り込んでいます。基本的に細かい絵が描けない人なので(笑)。たいていA4サイズのケント紙に描きやすい大きさで描き始めて，描ききれずにはみ出た部分は紙を追加していきます。今回は顔の部分(A)，上半身(B)，下半身(D)，パレオ(C)の部分の４分割です。
顔の部分がどうしても気に入った出来にならなかったので，ほかの作品に使おうと思っていた顔の下描きを身体の大きさにあわせて縮小して合成しています(だからちょっと不自然な感じ。しかも顔自体は左右の半分だけを描いてライトボックス上で反転コピーしたものだったりする)。最近は，ペン入れはせずに鉛筆画をそのまま下描きにしてます

なんとなくメカが描きたかったので，実験と称して分解したハードディスクの中身のメカ(ヘッドの部分あたり)を参照しながら適当な機械を描いてみました。以上３点が作品作りの素材として描いた下描きです。僕がCGを描くときは，こんな感じで人物と背景とアイテムを別の下描きで分割しておくことが多いです（ちなみに雲とか空とか草原とかを描くときは下描きはせずに直接タブレットに向かってます）

キャラクターをベースカラーで塗ってみます。色はあとで調整できるので，あまり細かいことは気にせずにだいたいの色調で塗ります。ちなみに印刷のことは考えていないので最初っからCMYKではなくRGBモードで描いています。線画を潰さないように，肌のレイヤーと服装のレイヤーをそれぞれ追加して鉛筆ツールの太いものでざくざくと描きます。細かいところは筆ツールで塗っていきます。服装のレイヤーが肌レイヤーより上にくるのはいうまでもないですね

さらに顔を塗ります。肌の陰になる部分を，ベースカラーよりも赤系に若干寄った彩度の高い色を使ってエアブラシで着色します。このときは「透明部分の保護」のチェックをしておいたほうが色がはみ出ないので便利かもしれません。髪の毛も別レイヤーを作成して，とりあえず単色でがしがしと塗っていきます

髪の毛の輪郭をていねいに仕上げていきます。前髪の生え際のあたりはエアブラシツールと筆ツールでグラデーションがかかるように滑らかに塗ります

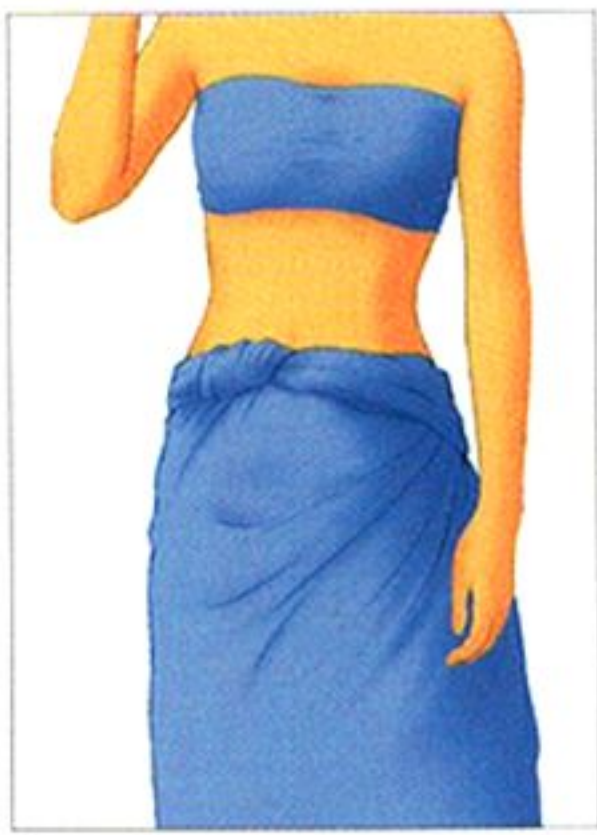
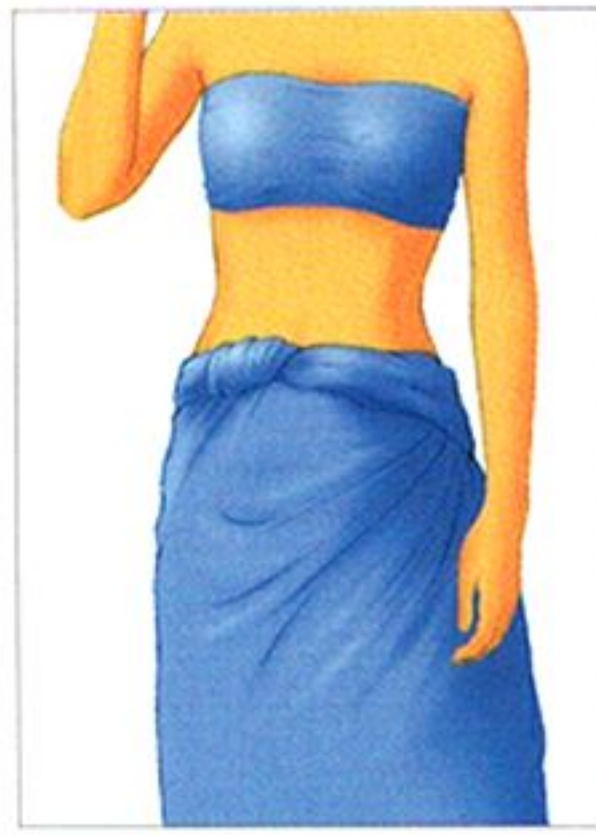
さらに極細の筆と指先ツールを用いてヘアスタイルを仕上げていきます。前髪は，おつむのてっぺんからの流れを意識して描きます。毛の穂先のほうが濃くなりすぎないように，ときどき消しゴムツールで調整します。この絵では，線画を茶色系に色調整し，瞳の部分も着色してみました

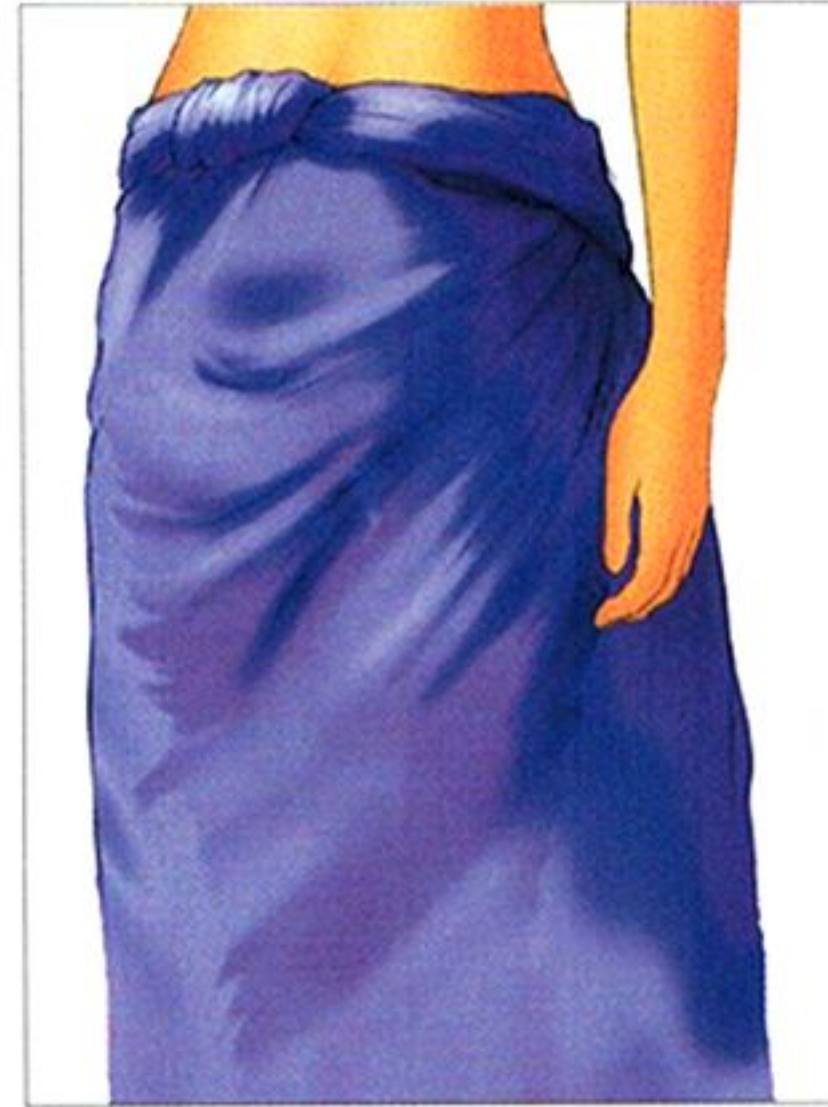
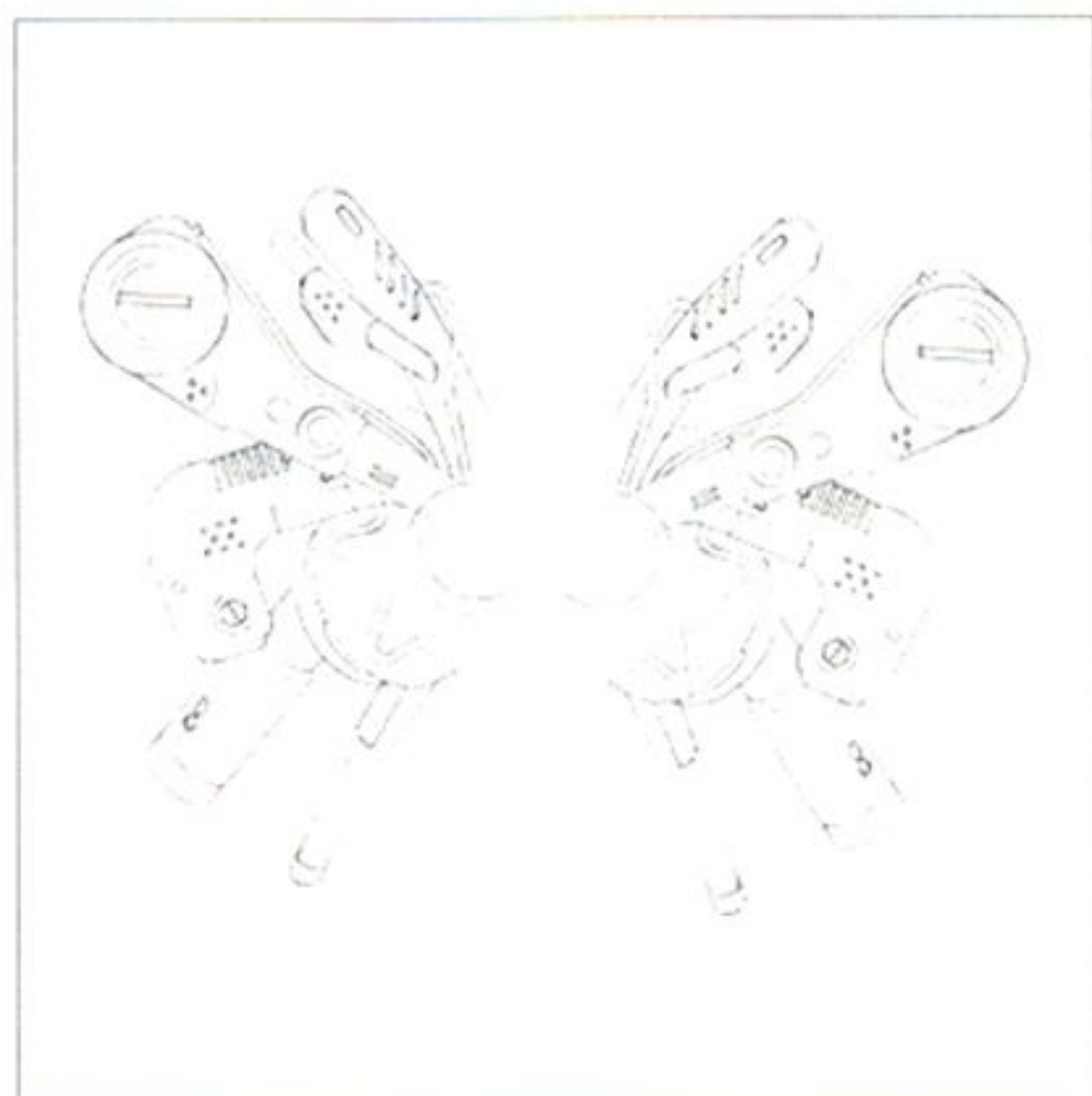
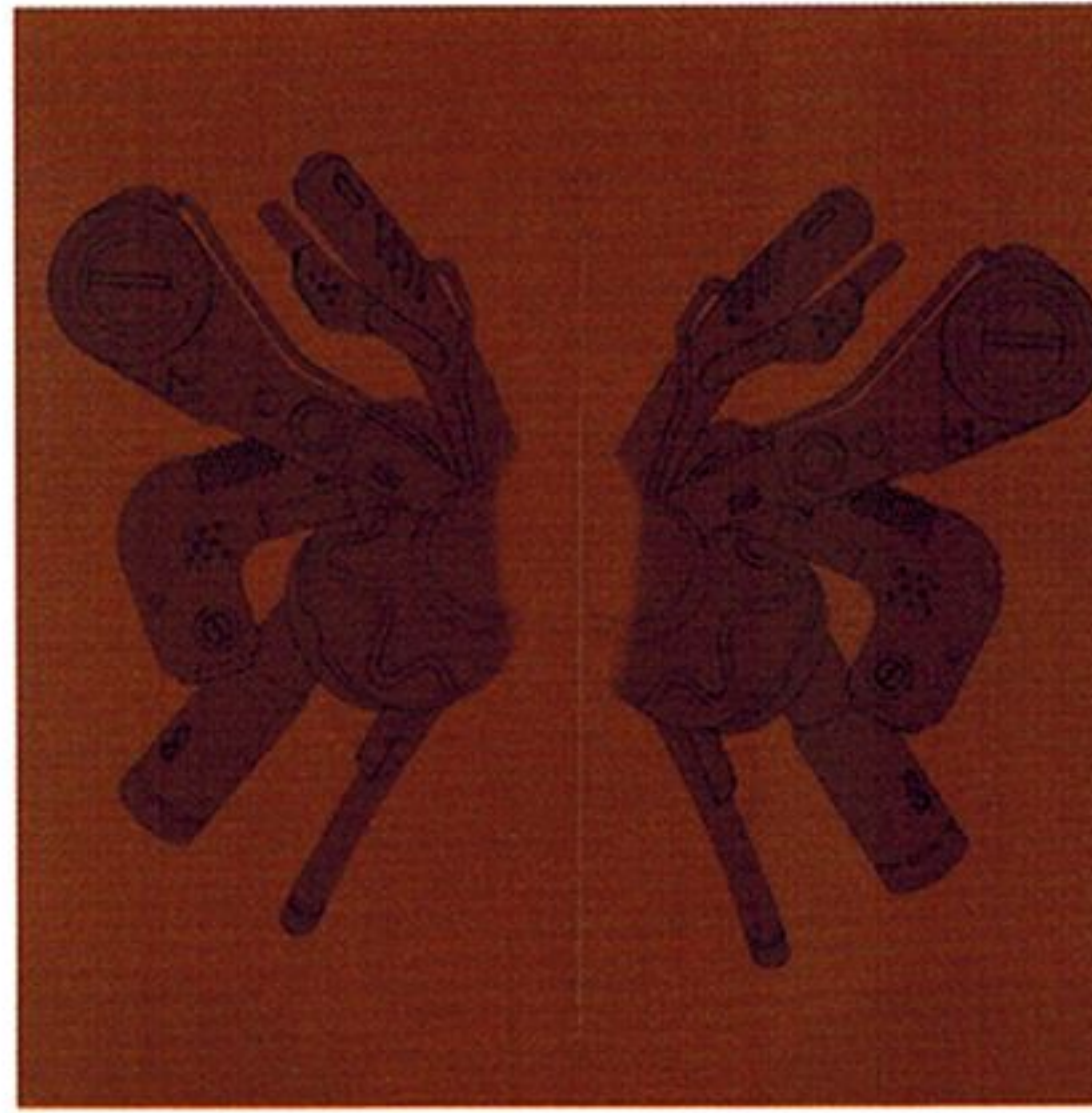
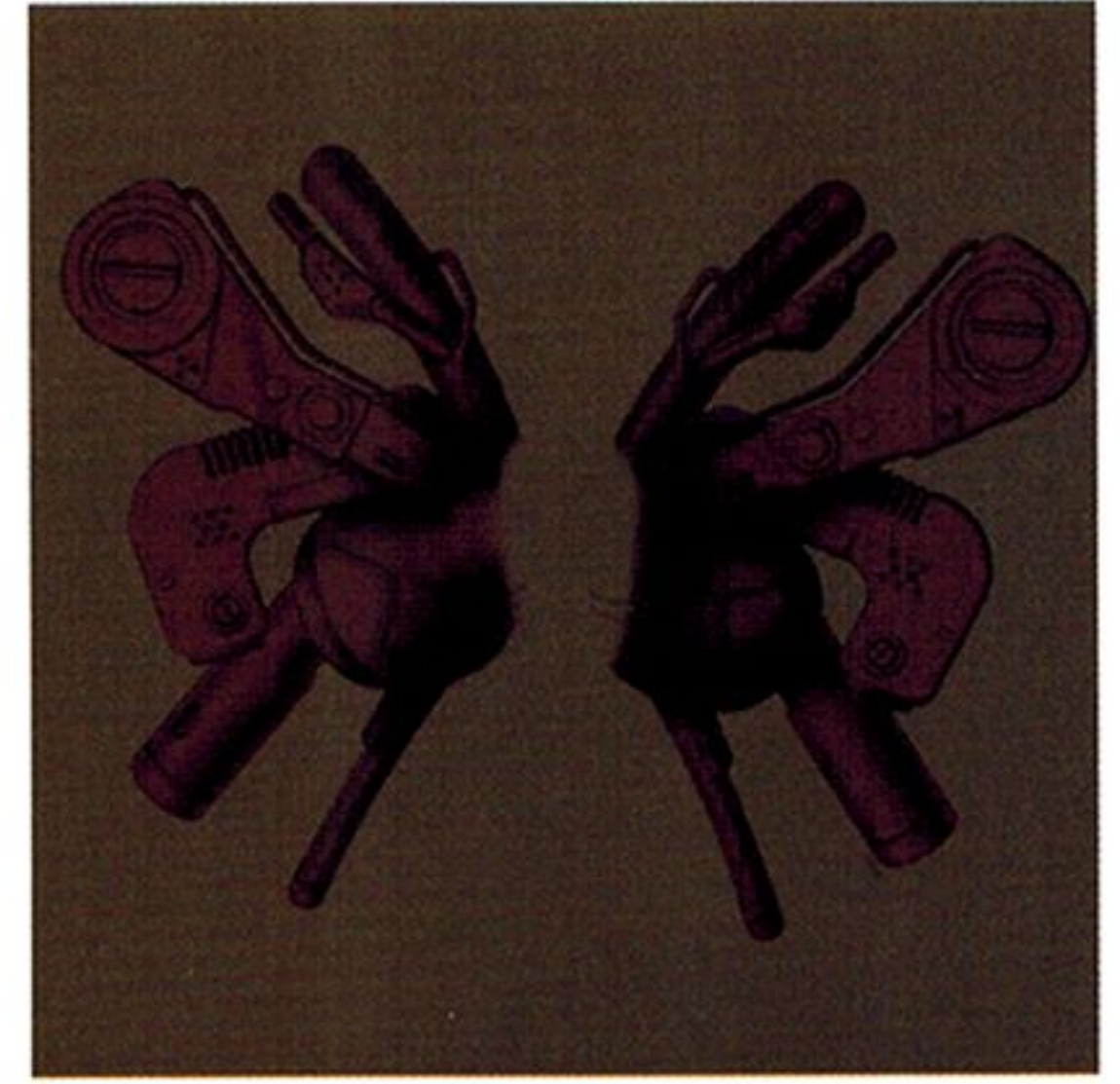
髪の毛の形状が整ったら，ハイライトを入れていきます。覆い焼きツールのハイライトモードを用いて，髪の毛の流れを意識しながら明るい部分をつけていきます。ストレートヘアなら「天使の輪」程度の加筆でよいかもしれません

ハイライトの流れを指先ツールで強調していきます

コントラストを調整してみます。髪の毛は，コントラストを強めにしたほうが綺麗になると思います。ただし，コントラストを強くすると彩度も上がる傾向があるので，不自然にならないように彩度を下げています

顔の立体感を考えて，エアブラシでハイライトを軽く入れていきます。このキャラクターでは，頬と鼻すじと唇のあたりを明るくしています。影付けのノウハウは，アニメ系のイラストを参考に研究するとよいと思うのですが僕は勉強不足です

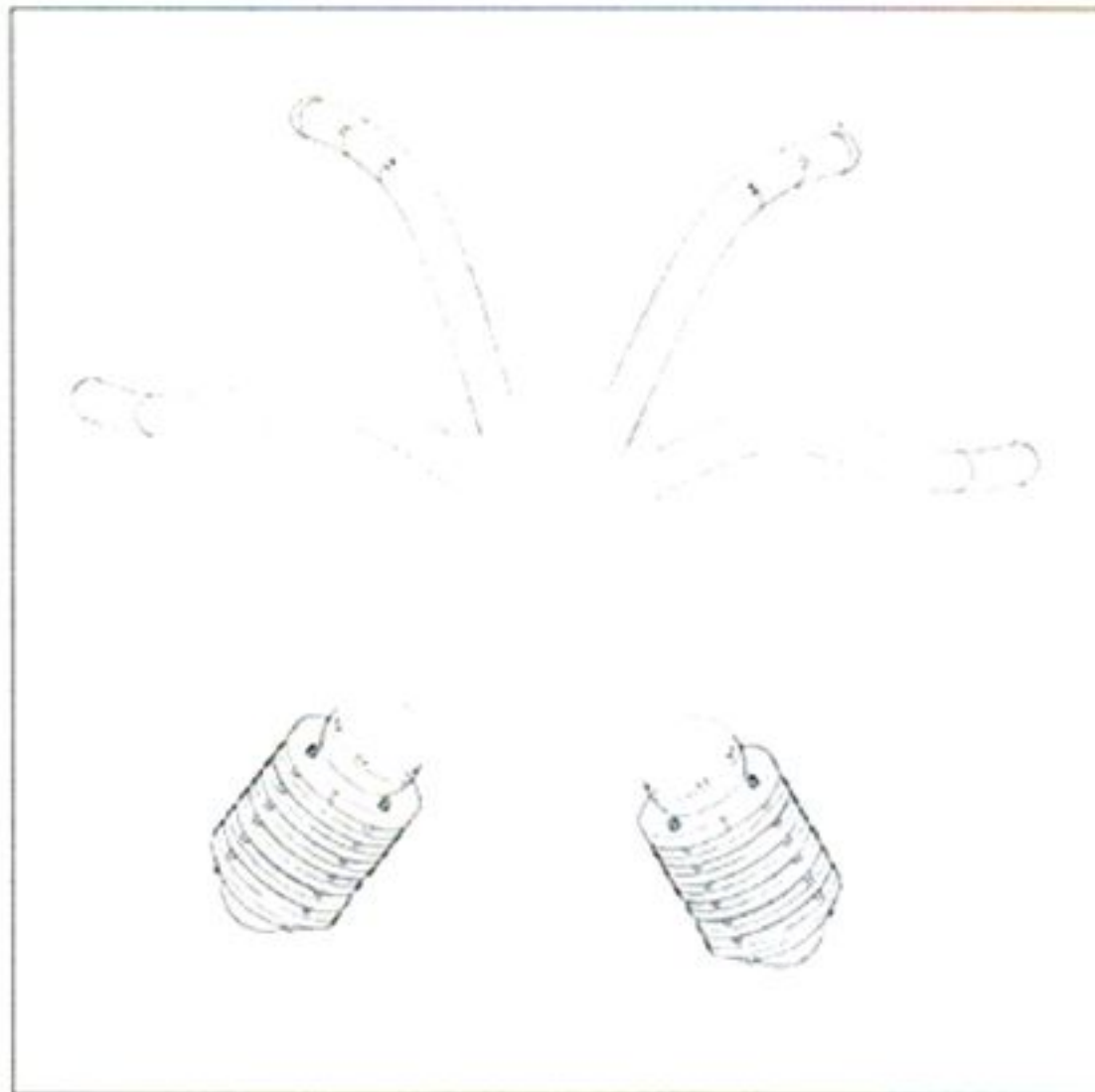
仕上げです。線画の眼や眉を肌レイヤーと合成し，指先ツールで眉毛やまつ毛の流れに沿ってこすって肌と馴らしていきます。僕の癖で，眉の流れが左右で違っちゃってるんですけど気にしないでください。頬紅を入れて陰を整えてとりあえず顔は終わり(実はあとで口紅を薄くひいてるんだけど)

服の陰影を入れたところ。エアブラシと指先ツールでだいたいの形を描き込んで，堅くなったところはボカシツールを入れています

ハイライトを入れたところ。今回は覆い焼きで入れてますね

遊びでトーンカーブを調整したら，思いがけずシルクのような輝きが出ました。これでいってみようと思います

スケベ心が出てきて，脚が透けて見えているようにしてみました（せっかく身体のほうも描いてるんだし……）。
　まずレイヤーオプションで身体のレイヤーが透過して見えるよう調整します。そのままだと布のしわや厚みの感じがなくなるので，指先ツールで脚のほうを布の流れにあわせて変形しています

メカの絵をコピー＆左右反転，回転移動などしてシンメトリーな構造にしています（立体感がねーなぁ，くそう……）

ベースカラーを塗ります。例によって，あとで色はいじくれるので，あまり深く考えずに適当な色で塗ります。背景色がころころ変わってますが，そのときの気分で変えているだけなので気にしないでください

陰を入れています。焼き込みツールを使う場合もあるし，明度を落として彩度を上げた色をエアブラシで塗り込むこともあります。金属っぽい雰囲気を狙うなら焼き込み＆覆い焼きツールのほうがうまくいきますね

というわけで，ハイライトを覆い焼きツールを使って表現したところ。簡単な操作でいい雰囲気が出るので，このあたりの作業は楽しいです。あまりに綺麗すぎると不自然なので，雲描画フィルタで作った影を加算して，色を揺らがせています

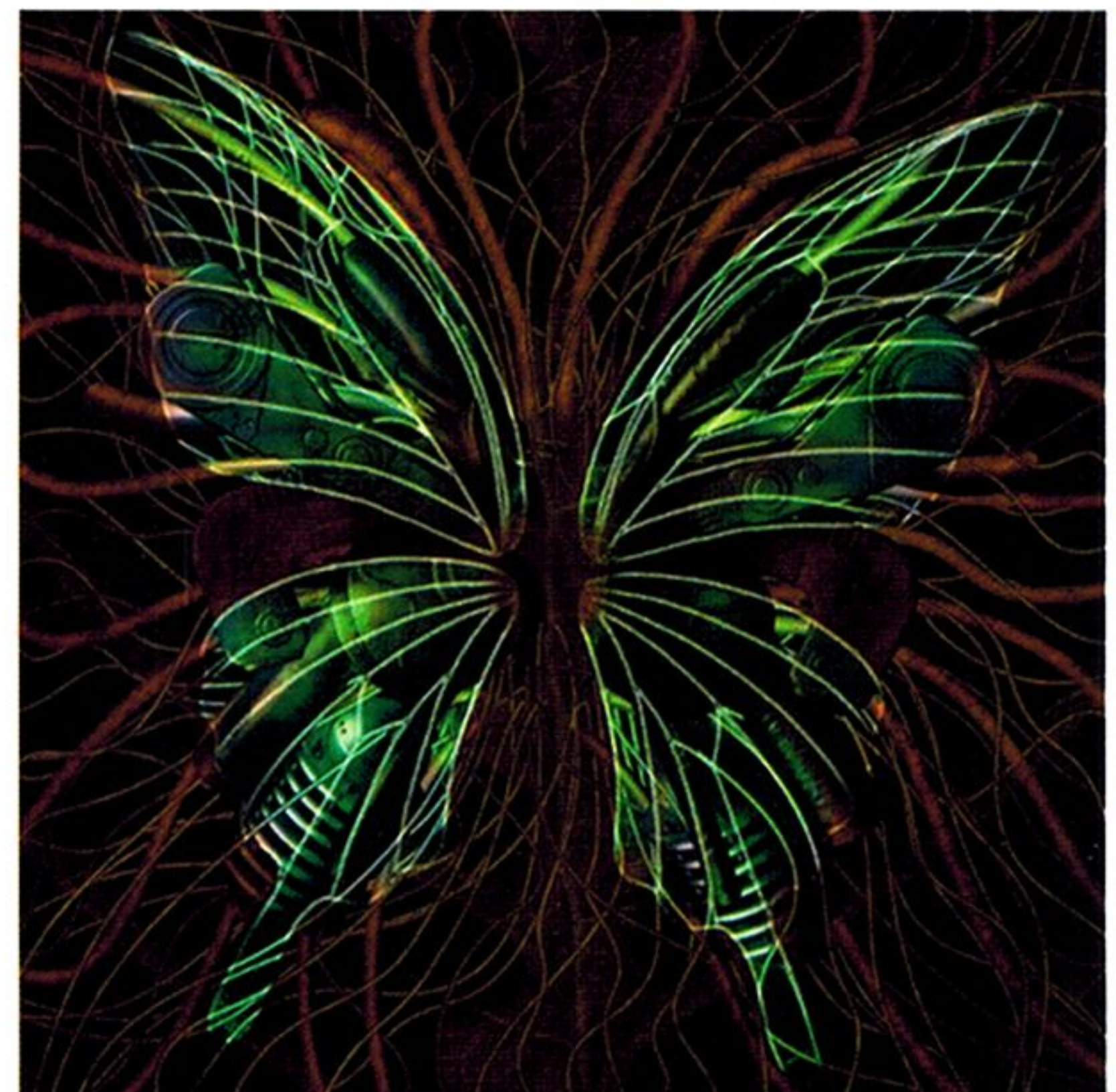
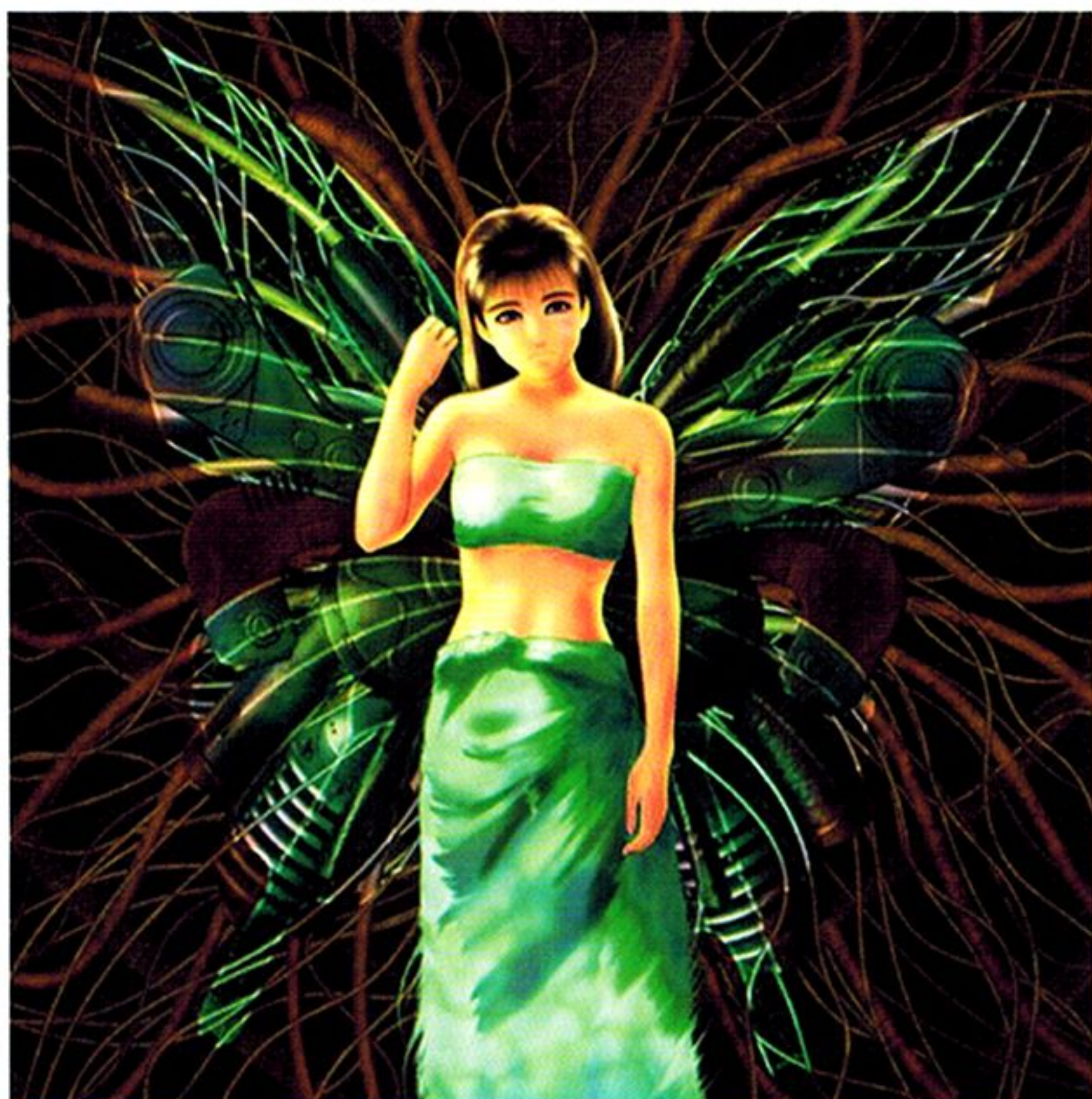
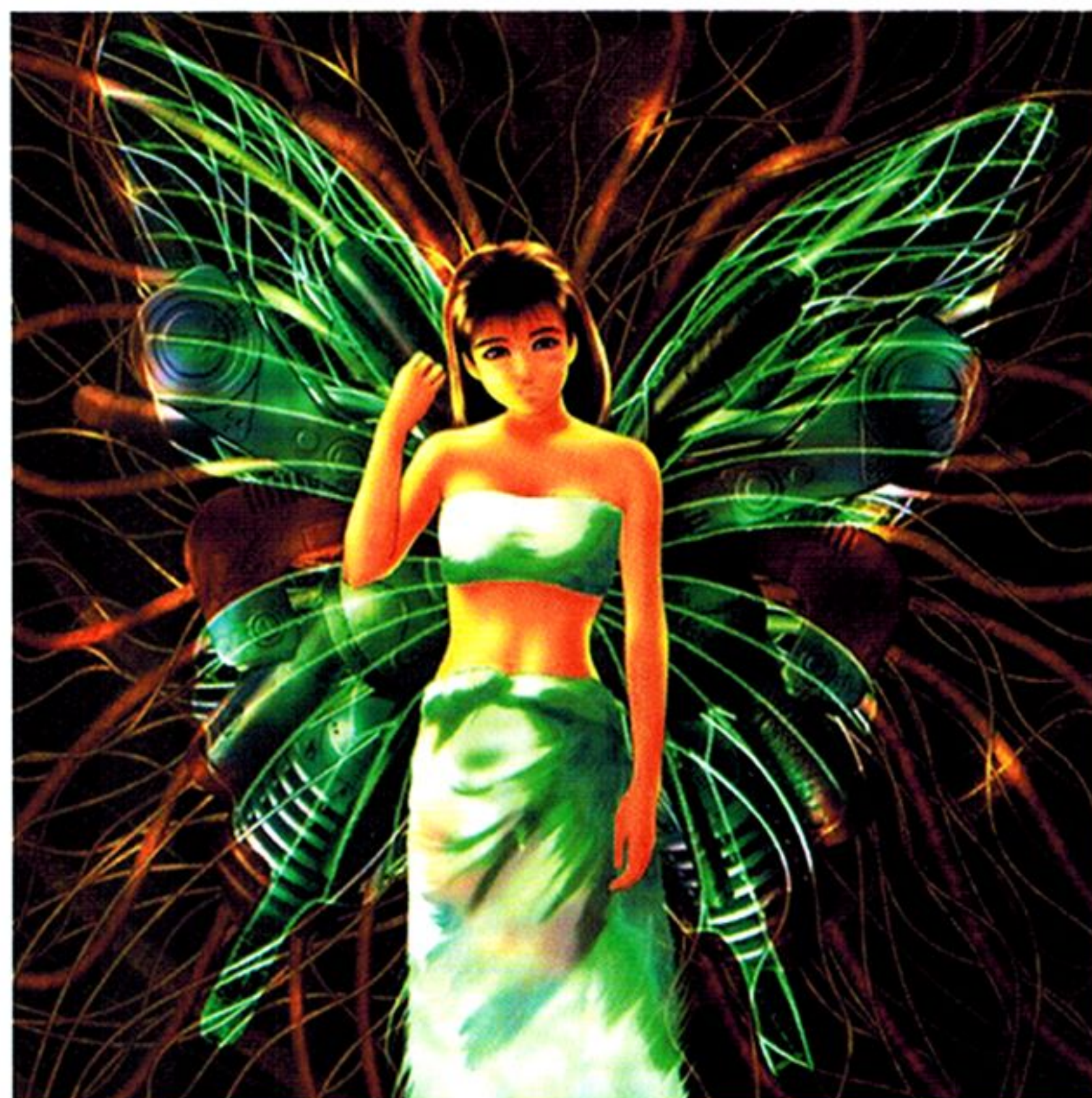
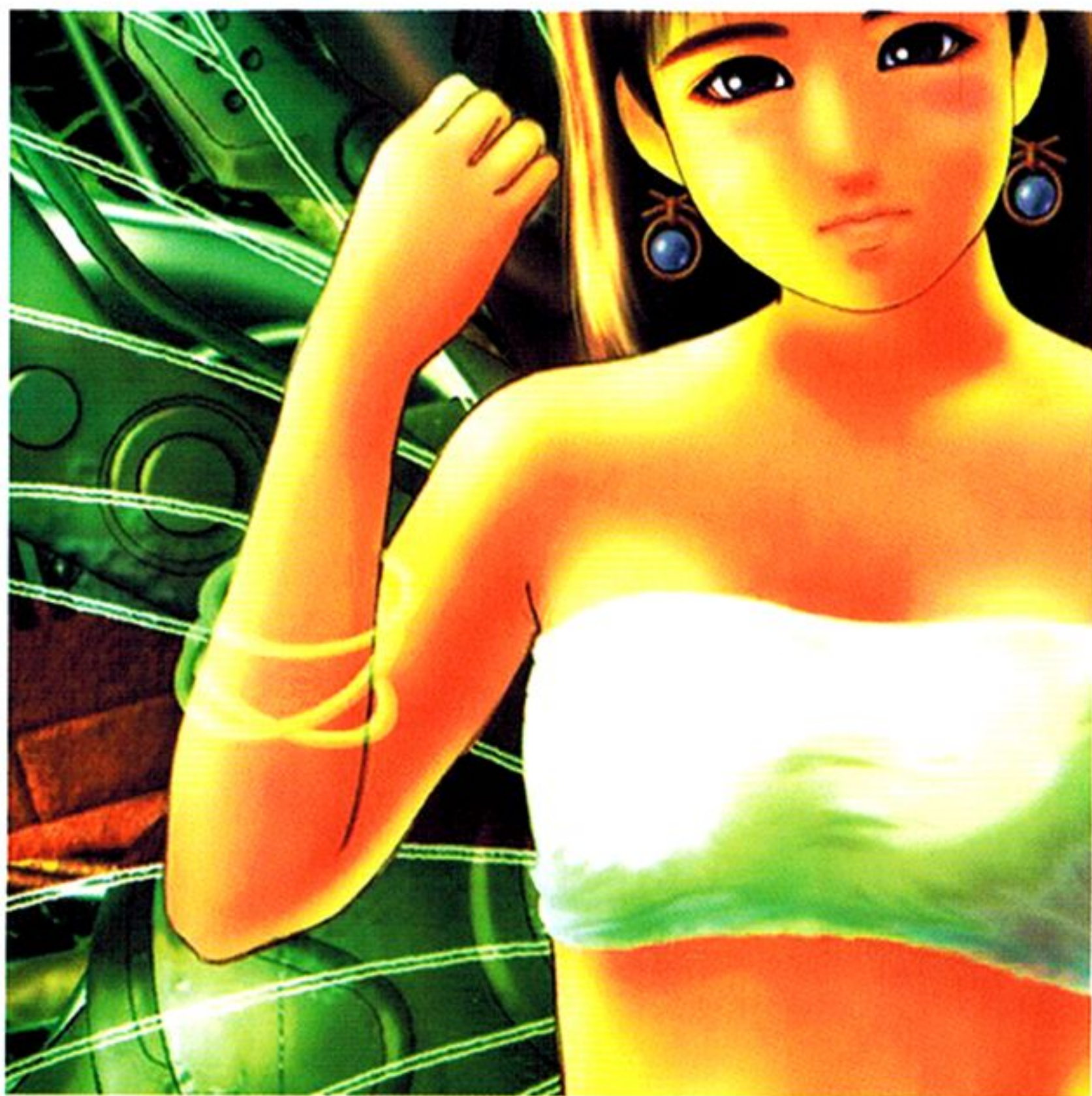
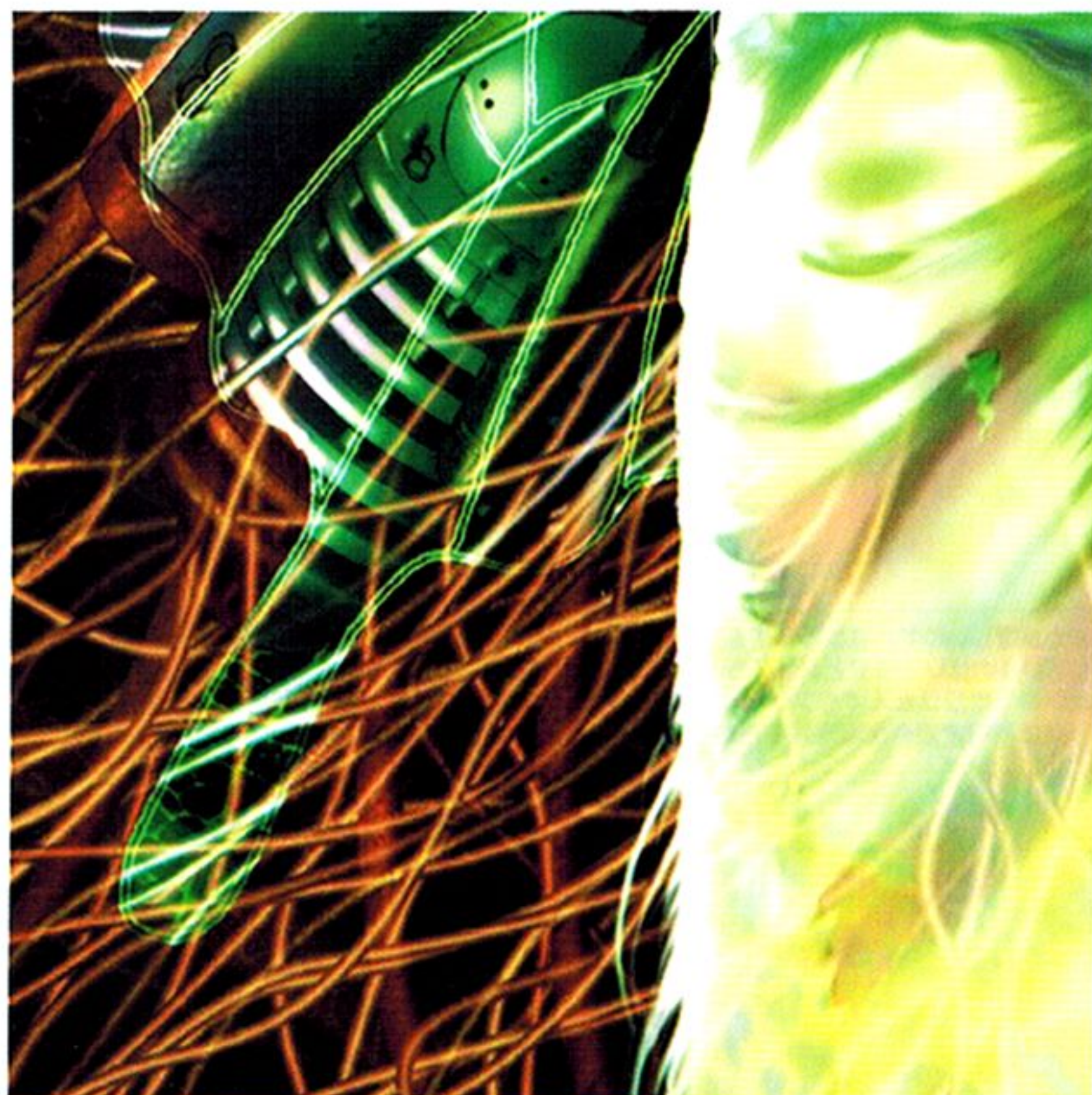
試しにキャラクターのレイヤーを載せて，色を調整してみたところ。角度や大きさの調整も，このあたりでやってしまいます

こっちは羽のレイヤーを載せてみたところ。ここで「画面がなんか寂しいなぁ」と思ったわけです

ワイヤーの類を描きます。こういったものは「イラストレーター」や「フリーハンド」などのドロー系ツールのほうが描きやすいのですが，僕はイラストレーターは持っていないのでPhotoshopのパスを使って描いています。たくさん描くのが面倒だったので，数本描いたら移動，回転，コピーして増殖させています。いちばん後ろのレイヤーは素材集CD-ROMから持ってきた岩の画像を調整したもの

メカが寂しいので，追加の下描き。片側だけ描いてコピー＆反転しているのはいままでどおり。ちなみに僕は定規の線が好きじゃないので普段はなんでもフリーハンドで描いてしまうのですが，今回は楕円定規や曲線定規を用いて真面目に描いてます

色を塗ったところ

背景の仕上げを始めたところ。羽を浮き立たせるために，羽の形に切り抜いた緑系のレイヤーを重ねて色を調整しています。もっとゴチャゴチャした感じにしたかったので素材集の木の葉の葉脈を岩とワイヤーの間に重ねています

当初のメカの後ろに配置してみました。少しはにぎやかになったかな

キャラクターのレイヤーを載せてみたところ。服装の色調を羽の色と統一させようという意識が出てきました

キャラクターの位置を微調整し，覆い焼きツールなどを駆使してコントラストを変更しています

アクセサリを追加。この類のデザインは苦手なので効果もイマイチ

足元が，やっぱり寂しく感じたのでワイヤーを追加

反省点

というわけで，どうもお粗末さまでした。指摘される前に反省点を書いてしまいます。蝶と娘というコンセプトが陳腐。全体的に立体感がない。光の方向があやふや。ポーズが堅くて変。特に右手の行動が意味不明。服装のライティングが不自然。目線は逸らしたかった。などなど……。次回作にはこういった反省を生かしたいですね。

おわりに

自慢じゃないですが，ここ数年は勤務先で購読してる日経ほげほげの類以外，ほとんどパソコン関係の雑誌をチェックしていませんでした。で，先日，本屋のパソコン雑誌コーナーに立ち寄って驚いたことといえば，3D，2D，DTPも含めたCG関係の雑誌の多さでした。何種類もの月刊，隔月刊の専門誌が一般の書店に平積みされているのを見て，こんなに需要があるのかねぇと，かなり不思議に思いましたが，それなりに売れる見込みがあるから出版してるんでしょうね。

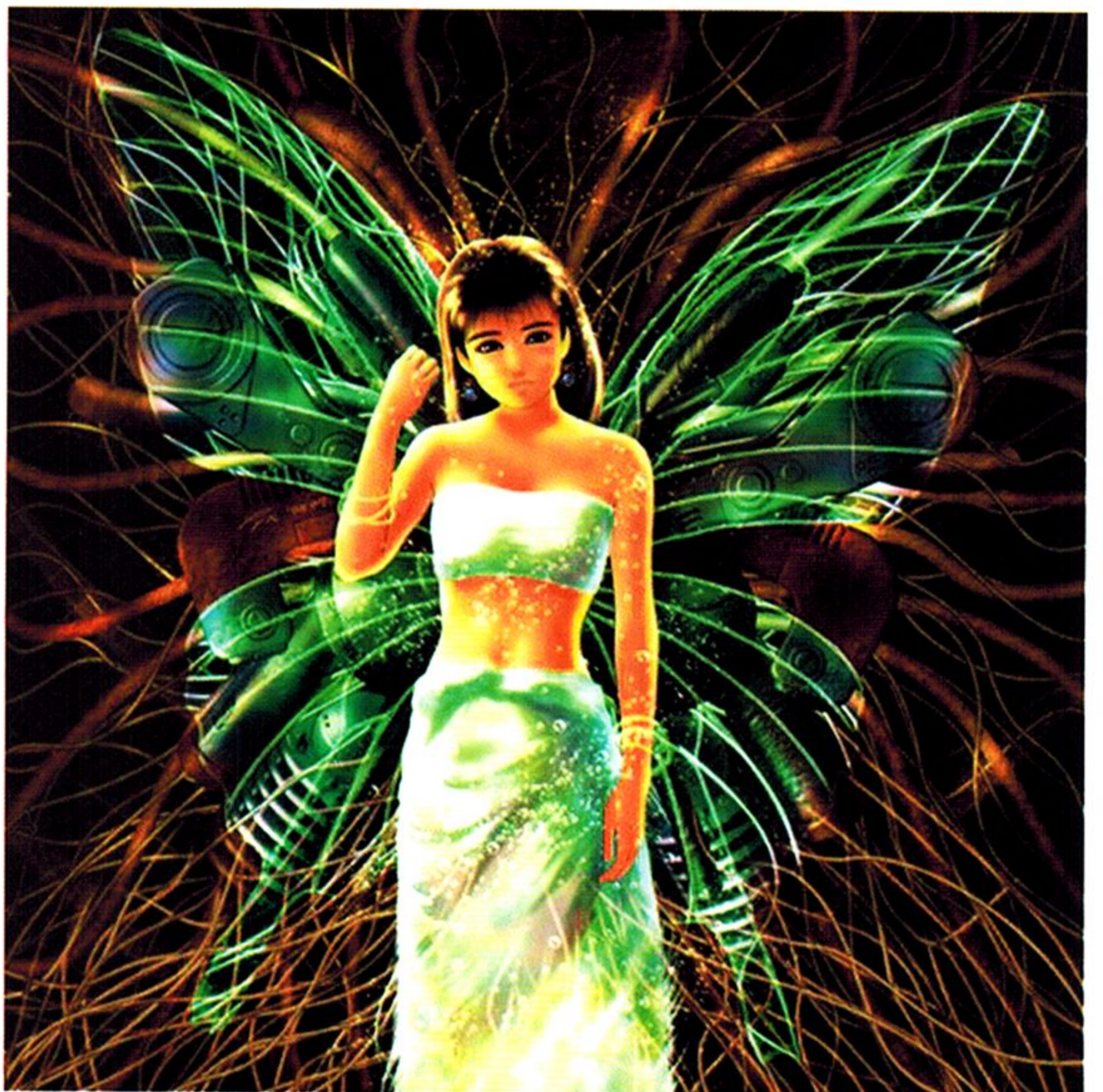
一時期のパソコン雑誌創刊ラッシュの顛末のようにやがては自然淘汰されていくのでしょうけれど，ホームユースパソコンのひとつの楽しみ方として，また趣味でマンガを描く人口の多さから考えてCGがいよいよ注目されているということはいえると思います(プログラミングに比べて敷居が低いですし)。

気泡を浮かべてみました(やめたほうがよかったかも)。この気泡だけはPainterを使って描いています。自家製の泡データをノズルを使って散らし，そのレイヤーをPhotoshopに持ってきて合成しています。モロ都築和彦先生の影響ですね

これでだいたい完成なのですが……

ただ，ぱらぱらと雑誌をめくってみたのですが，イマイチ「使える」と思わせられるものや発見のヨロコビを感じさせてくれるものはありませんでした。好みの問題もあるのかもしれませんが，素材集やフィルタやフォントに頼っていてなにかのっぺりとした没個性の作品が多いと思っています。偉そうにいっていると思われるかもしれませんが，受け売りではなく正直な感想です。CGは珍重されていた時代から，むしろ卑下される時代になりつつあるのではないかというのが実感だったりします。個性とアイデアの追求がますます大切になりますね。以上，自戒の念も含めて。

繰り返しになるかもしれませんが，絵的センスや工夫がなくても，素材集やフィルタの組み合わせでそれなりの絵がなんとなくできてしまうのが2D CGの面白いところであり，落とし穴でもあります。自己満足で閉じてしまうのならツールに頼るのもよいのですが，ほかの人にも見て喜んでもらえる作品作りには情熱と個性が絶対に必要だということを最近つくづく感じています(あ，つまり自分のことです)。

もはや環境が貧弱だから，といった甘えは許されません。プロであれアマチュアであれ，個人の能力を出しきれるインフラは幸か不幸かすでに十分用意されてしまっています。あとは使う側次第。いずれにせよ，誰もが安価にCGを楽しめるようになったのはよいこと，というところでまとめとしておきます。

まぁいろいろと不満もあるんですが，一応これで完成とした。サインを入れたりして(ちょっと気取って右肘の横のメカのあたり)作業終了

はいぼく

h a i b o k u

縁なしめがねを描く

使用環境
本体：PC-9821 V200(Pentium-200)
メモリ：96M
HD：内蔵2GB，外付け4GB
MO：230MB
スキャナ：Canon，CanoScan300(300dpi)
タブレット：WACOM,DIGITIZER Ⅱ
ソフト：Photoshop,Painter5など

はじめに

どんな女の子でも，めがねをかけるだけで少なくとも3割増しでかわいく見えます。ということで，女の子はどんどんめがねをかけるべきです。ここでは，女の子に縁なしめがねをかけさせる方法を解説します。

基礎知識

めがねを描くとき，デッサンが狂ってしまうことが往々にしてあります。ツルとレンズは直角に交わっていなければならないのですが，なかなかうまく描けないものです。このめがねのデッサンは，プロでも間違えている人が非常にたくさんい

紙の上にだいたいのイメージを鉛筆で下描きをしていく。この場合は4サイズの紙に描いている

先ほどの鉛筆描きの下描きをトレースして，ちゃんとペン入れし，それをスキャナでパソコンに取り込む。なお，この時点ではキャラクターだけ，それもめがねなしの状態で処理を進めていく

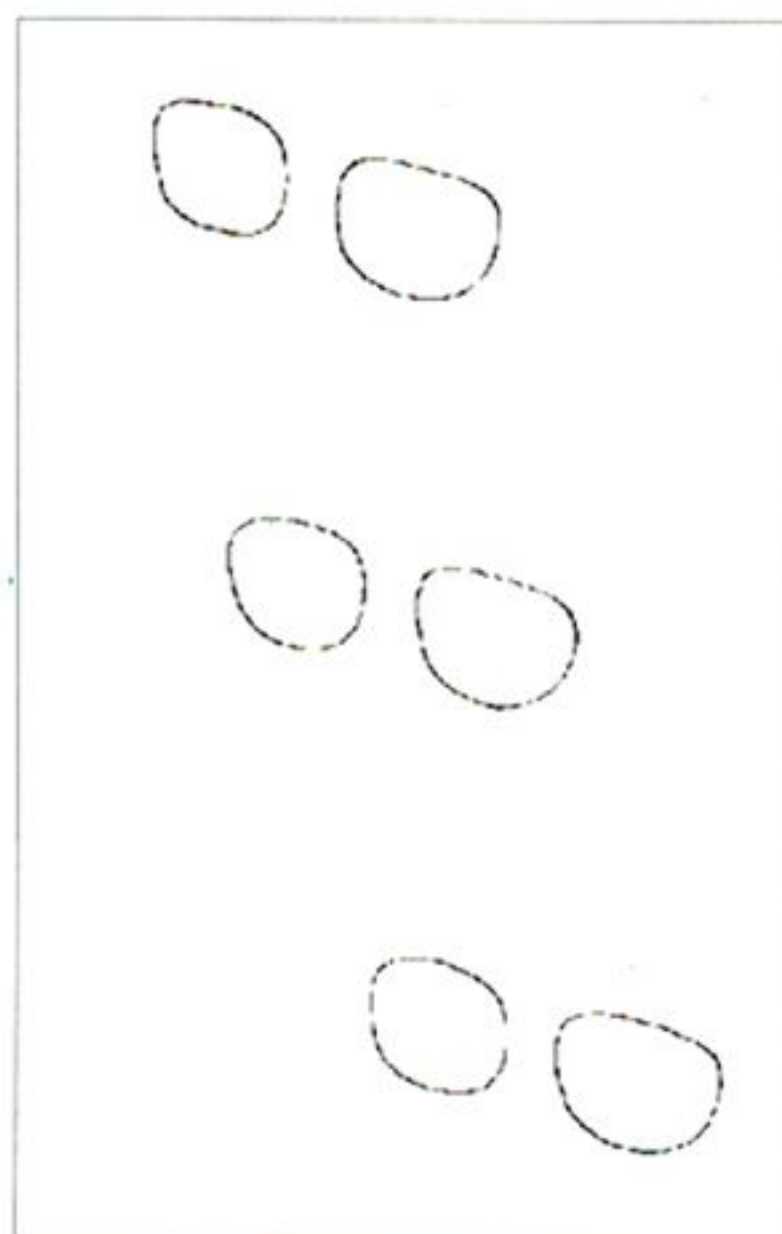
ます。ただし，マンガの場合，デフォルメをしてわざとデッサンを崩す場合がありますので，各人のデフォルメの仕方にマッチしためがねの描き方を修得する必要があります。顔のデッサンをわざと崩した場合，それにあわせてめがねのデッサンも変えなければなりません。めがねのデッサンに関する基本的な考え方はわしのホームページ(http://www.bekkoame.ne.jp/ro/haiboku/)のめがね描き方講座にありますので，参照してください。

コンピュータの威力

めがねを描くうえで，コンピュータは非常に優れた画材となります。まず，拡大縮小や回転など，画像を簡単に加工することができるので，紙に描いた時点で狂っていたデッサンも，コンピュータ上で修正することができます。そして，これまではエアブラシなどを駆使しなければ描けなかったツーポイントのめがねが，コンピュータ上ではいとも簡単に描くことができます。

縁なしめがねの描き方

⓪ソフトはWindows版のPhotoshopを使用しています。基本的な使用法を理解しているものと前提して解説します。

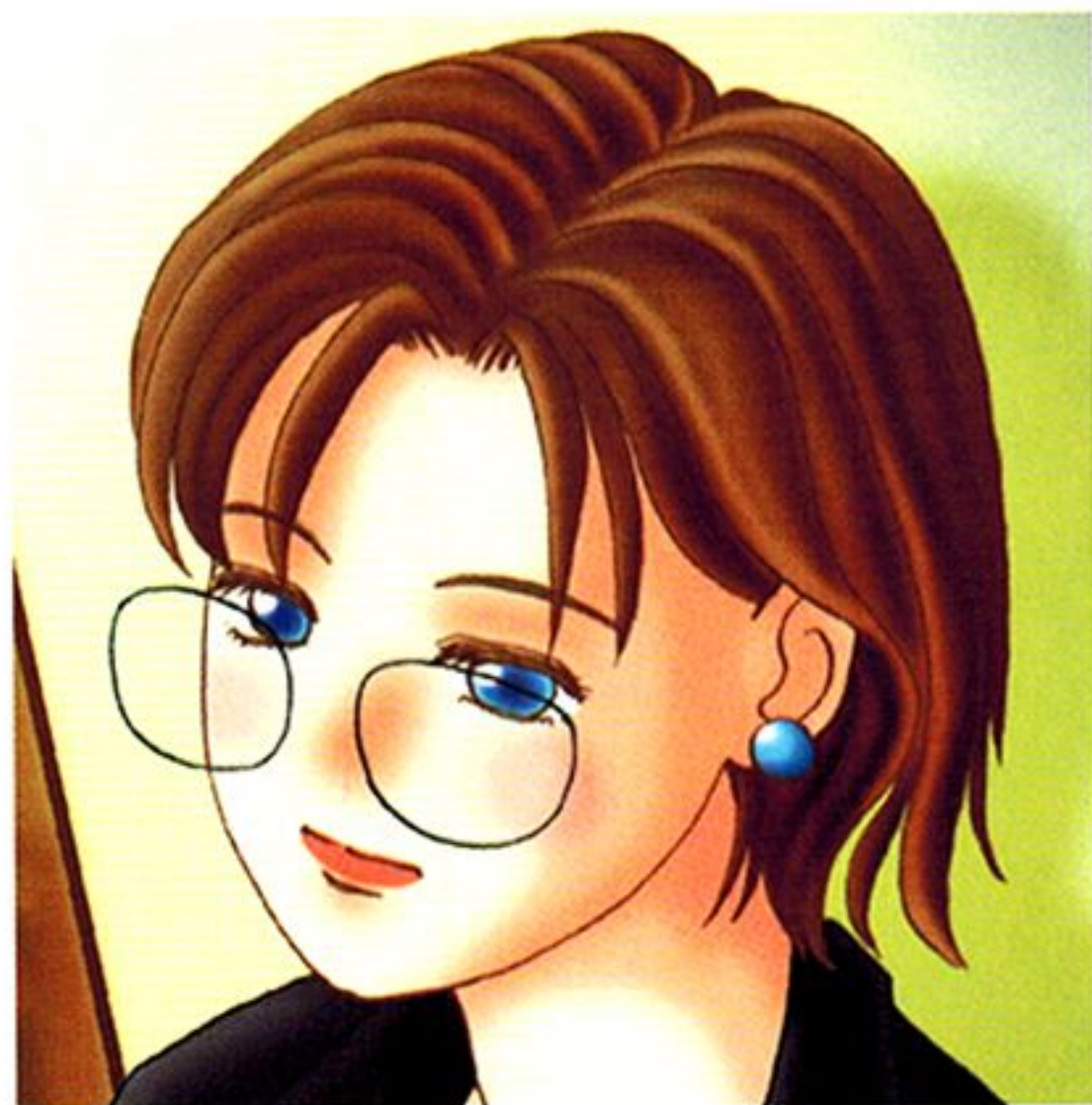
①まず，めがねの似合いそうな女の子を描きます。わしは，紙に下描きし，それをトレースしてペン入れしたうえで，スキャナで読み込んでいます。縁なしを描く場合，下描きにはめがねを描いておきますが，ペン入れの段階ではめがねは描き入れません。めがねがないのはなんか寂しい感じがしますが，一時の辛抱です。すぐにめがねをかけさせてあげましょう。

メガネレンズ部分は別途トレースのうえスキャンしておく

一気に説明は飛ぶが，まず，メガネ以外の部分を完成させておく

②そして，めがねの輪郭もペン入れして，スキャナで読み込みます。

③ちゃっちゃと人物を彩色し，背景も入れてしまいましょう。あとはめがねを描くだけにします。画竜点睛を欠いているのでまぬけな画像ですが，これから命が吹き込まれます。

④別に取り込んでおいためがねの輪郭を，新規レイヤーにコピーし，このレイヤーを乗算モードにします。移動ツールで，めがねを似合う位置に移動させます。ここでデッサンの狂いが発見されたら，「レイヤー自由変形」などで調整します。

⑤新規レイヤーを作成し，作業用のマスクを作ります。このマスクは，レンズの側面や陰影を描くときにひっきりなしに使うことになります。まず，先ほどの「めがねの輪郭」を選択範囲に

まず，レンズの位置を顔にあわせる

レンズ面をマスクにコピーし，レンズの後面マスクを作成する

光源を意識しながらレンズの側面を描き込んでいく

つるや金具をガシガシと描き入れる

最後の仕上げ段階。金属フレームの微妙な光沢や陰影を仕上げ，細かいパーツも描き込んでいく左の絵だと髪の毛などとの優先順位もいい加減だったのだが，消しゴムツールでその辺の辻褄をあわせていく，これで完成だ

めがねだけだとこうなる

して，新規レイヤーを黒で塗りつぶします。まるでサングラスみたいですが，ただのマスクなので，最後には消去します。このレイヤーは「レンズ前面マスク」とでも名づけておきます。このレイヤーをコピーして「レンズ後面マスク」と名づけます。そして，前面マスク，後面マスクのレイヤーの透明度を40％程度に下げ，下にある画面が透けて見えるようにします。続いて，レンズの前面，後面に対応するように移動させます。

⑥さて，マスクの用意ができたところで，レンズの側面を描きます。レンズの側面は，光源の位置に近いところは暗く，遠いところは明るくなります。

では，先ほど作ったマスクを利用して，レンズ側面の選択範囲を作成します。まず，「レンズ後面マスク」レイヤーを選択しておきます。続いて，CTRLキーを押しながら「レンズ前面マスク」レイヤーをクリックすると，それが選択範囲となります。

選択ツールにして，GRPH（ALT）キーを押しながら画面上の「レンズ後面マスク」をクリックすると，その部分が選択範囲から引かれます。すると，レンズの光源側の側面が選択された格好になります。光源側は暗くなるので，濃い肌色で塗りつぶします。

同様に，光源から遠いほうのレンズ側面の選択範囲を作成し，白で塗りつぶします。ここでは，光源側の側面と遠い側の側面は別レイヤーで作ってあります。透明感を出すためにレイヤーの透明度を調節するので，暗いほうと明るいほうは別々のレイヤーにしておいたほうがいいわけです。

⑦続いて，レンズの影と光を描き込みます。先ほど作ったマスクを利用してレンズ内を選択範囲とし，描き込みます。影と光も別レイヤーで作ります。影や光が強すぎると思った場合，そのレイヤーの透明度を下げるだけで調節できますから，便利です。

⑧レンズが終わったら，金具をガシガシ描き込みます。わしは，タブレットで直接描き込んでいます。陰影は覆い焼きツール，焼き込みツールでつけています。ツルは，右側を描いたら，それを縮小コピーして左側に平行移動します。コピーを活用すると手間が省けて便利です。ここで，レンズの形がちょっと変わってるのに気がつきましたか？　ちょっとおかしな感じがしたので，レイヤー自由変形を使って調整したのです。

⑨見えない部分を消しゴムツールで消して完成。最後に金具の光沢やハイライトを入れるなど，細かい仕上げをして，画像全体が完成です。ちなみに，めがねだけだと，こんな感じになっています。

おわりに

女の子の絵を描き終わったとき，なにかもの足りないと感じたら，確実に「めがね」が足りません。描き加えましょう。そして，世界中の女の子をめがねっこにしてあげましょう！

森川久志

Hisashi Morikawa

椅子の少女

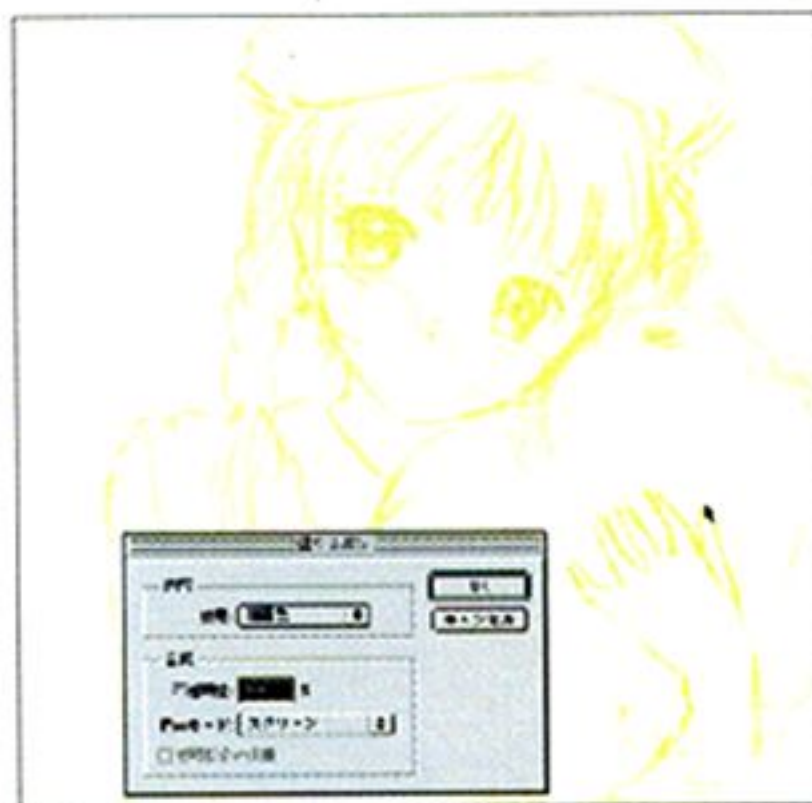
使用環境
PowerMac 7600/132 + メモリ96MB
スキャナ HP ScanJet2CX
タブレット WACOM型番忘れました
ArtPad2proの前身

人物の制作

人物はPainterのペンと水彩でひたすら描きます。Painterのスムースインクペンのサイズを小さくして±サイズを大きくとると漫画に使うGペンらしくなります。手法も重ね塗りと塗りつぶしの2種類用意すると便利です。

鉛筆描きの下絵をスキャンして取り込みます(画像1)。

Photoshop上で下絵の線の色を薄い色に変えます。具体的には下絵の線を識別できるぎりぎりの色を選択し,「塗りつぶし」→「選択色で塗りつぶす」→「スクリーン」を実行(画像2)。その状態を保存して,Painter上で下絵の薄い線の上から黒でペン入れ。再びPhotoshop上に持ってきて「色調補正」→「レベル補正」のスポイトツール(白)で薄い下絵の黄色を選択して飛ばします(画像3)。

画像1　まずは下描きをスキャナで取り込む

画像2　下絵はできるだけ薄い色に変換して保存する

画像3　Painter上で下描きをもとにペン入れを行う

画像4　色変換して下絵の色を飛ばすと主線だけを取り出すことができる

画像5　色つきになるように指定するとこんな感じになる

画像6　主線の上にそのまま色を塗っていく。いちいちマスクは使わない

画像7　主線だけを取り出して合成してやるという手もある

画像8　肌の部分は「水彩」を使って塗っていく

画像9　服の部分などは「筆」を選んで厚く塗っていく

画像10　眼鏡は取り外しもできるように別レイヤーで作っておく

もちろん厳密には主線にも多少影響が出ますがこの際気にしません。

普通はこの状態(画像4)からグレースケールに変換したり,「色相・彩度」で彩度を落としたりして黒線に戻しますが,彩色のときのことを考えて主線にあらかじめ色をつけておくのもよいでしょう。この場合そのまま「色相・彩度」で主線を褐色がかった色にすればカラーインクのような味を出すこともできます(画像5)。

実は上記のような手順は下絵をPainterに読み込んでトレーシングペーパーの機能を使ってトレースすればよいのですが,下絵をそのままPainter上に読み込んでトレース機能を使うと主線の透明度も半透明になるため下絵の線と区別がつきにくくなって非常にやりづらいのです。そのために上記の手段を採用したのですが,Painterのトレーシングペーパーに透明度の調節機能さえあれば,なにも上記のような手間をかける必要などないのですが……。

人物に色を塗るときには,いちいちマスキングするようなことはしません。マスクが重宝するからといってどこもかしこもマスキングしていたら結果的に作業が面倒になります。最低限の範囲だけ投げ縄ツールで選択して,はみ出した部分は「水彩用消しゴム」であとから消します(画像6)。

では「水彩」以外の筆で塗る場合の主線の保護はどうすればいいかというと,Photoshopでは,白地の上に描いた主線画を別レイヤーにして合成に「乗算」を選択する方法や,主線画をレイヤー化してフリーウェアのプラグイン「Eliminate White」を使用したあと,「白マット削除」を実行し,主線部分のみを抽出する方法などがあります(画像7)。

Painterでは,主線を保護するのに「マスク」メニューの「自動マスク」を利用する方法などがありますが,この方法で作成したマスクを選択範囲として読み込んでも主線の薄いところはやはりつぶれやすいので,着色の際の透明度を下げて薄く塗り重ねるのがよいでしょう。つぶれた分は同じマスクを使って上から描き足すという手もあります。

この絵の場合はさほど主線を残すことにこだわらなくていいので最初から割と思い切って色をつけていきました。

肌の部分の水彩に対して(画像8),服の部分は厚ぼったさを出すためにPainterの「筆」カテゴリーの筆で描いています(画像9)。このカテゴリーの筆は質感を出しやすいので水彩の次によく使います。中でもキャメルブラシ,はけ状ブラシ,カバーブラシ,極彩ウォッシュブラシ,すじ状ブラシ,ローディッドオイルなどがおすすめです。レースの部分にはすじ状ブラシを使用。これはカーテンのレースや観葉植物の葉にも応用しています。

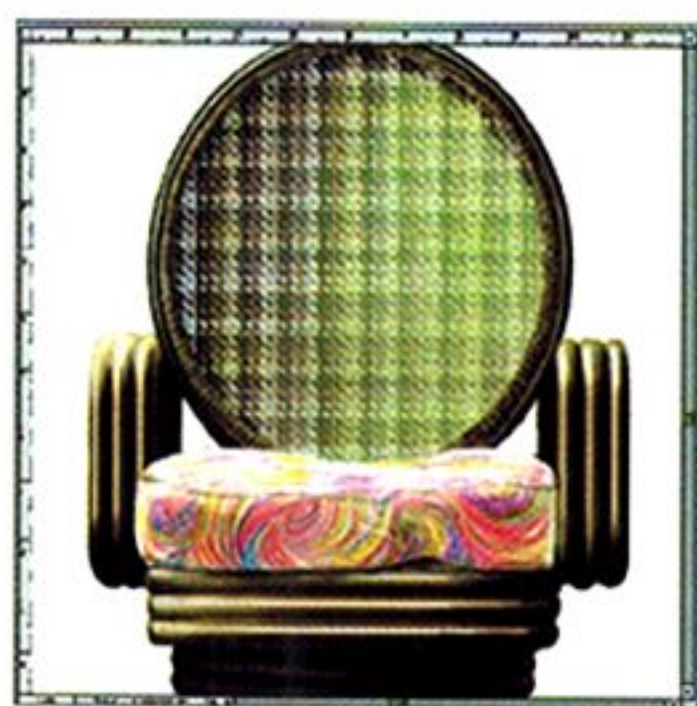
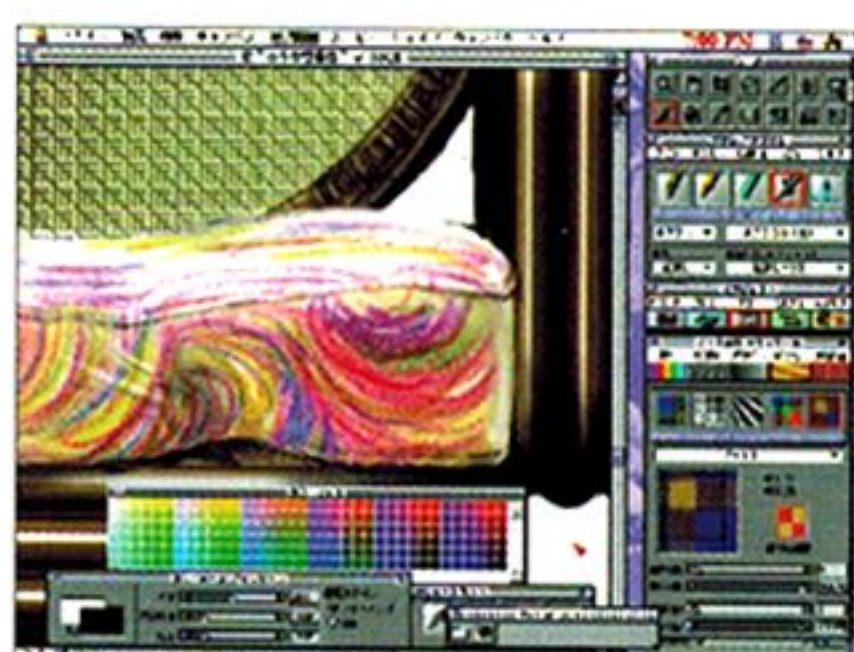
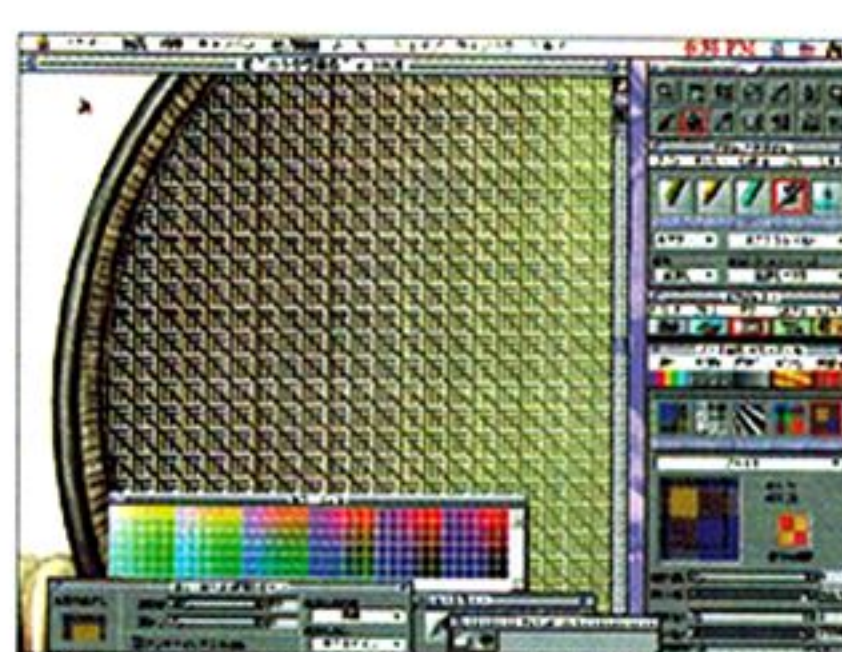
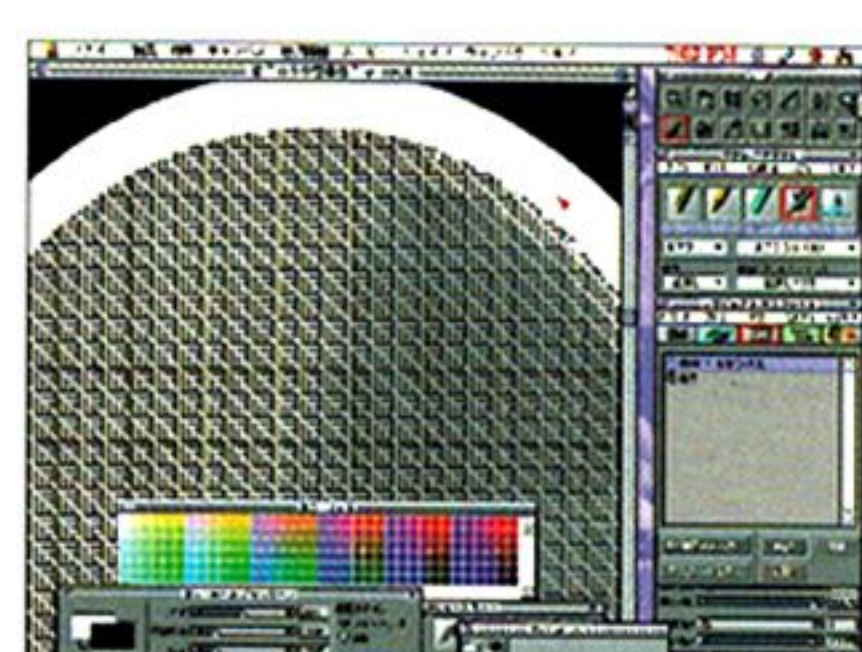
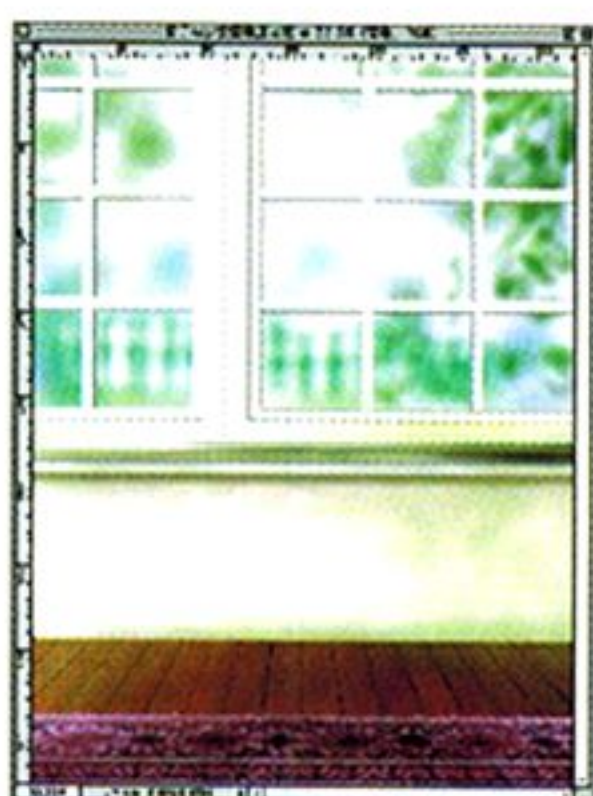
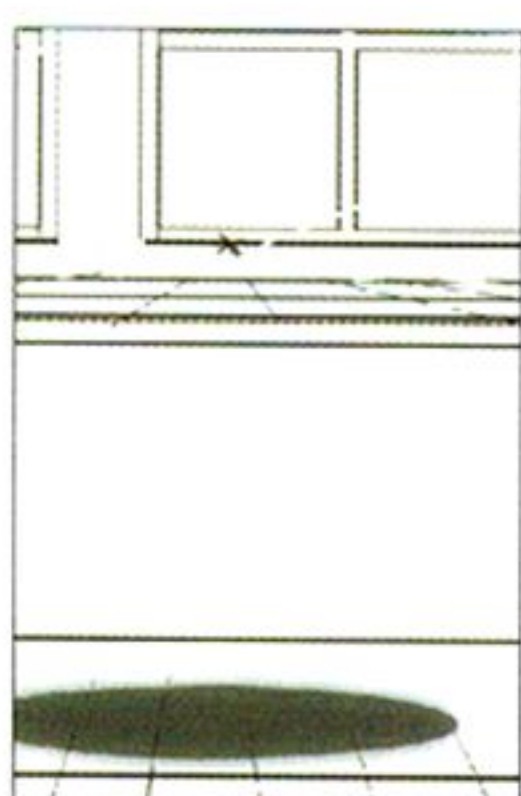
眼鏡に関しては着脱可能なように,あとからPhotoshopのレイヤー上に楕円選択ツールなどを使用して描いています(画像10)。

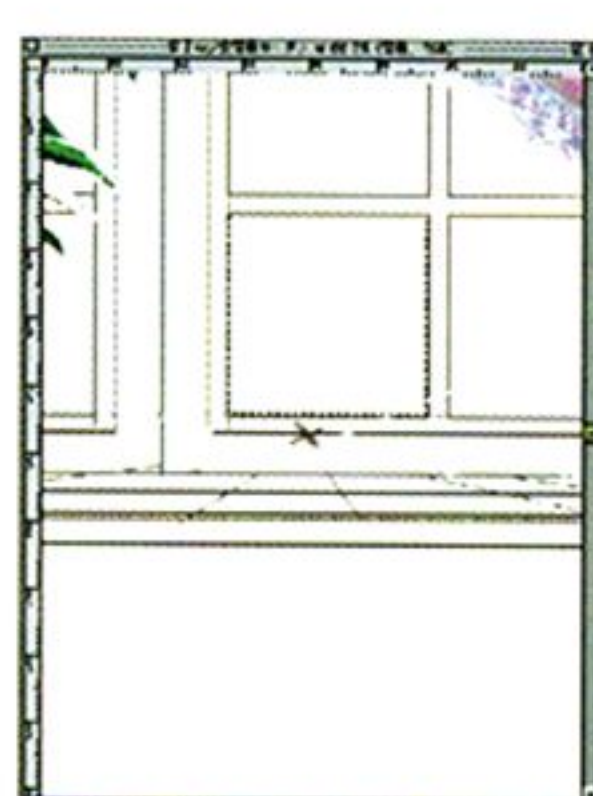
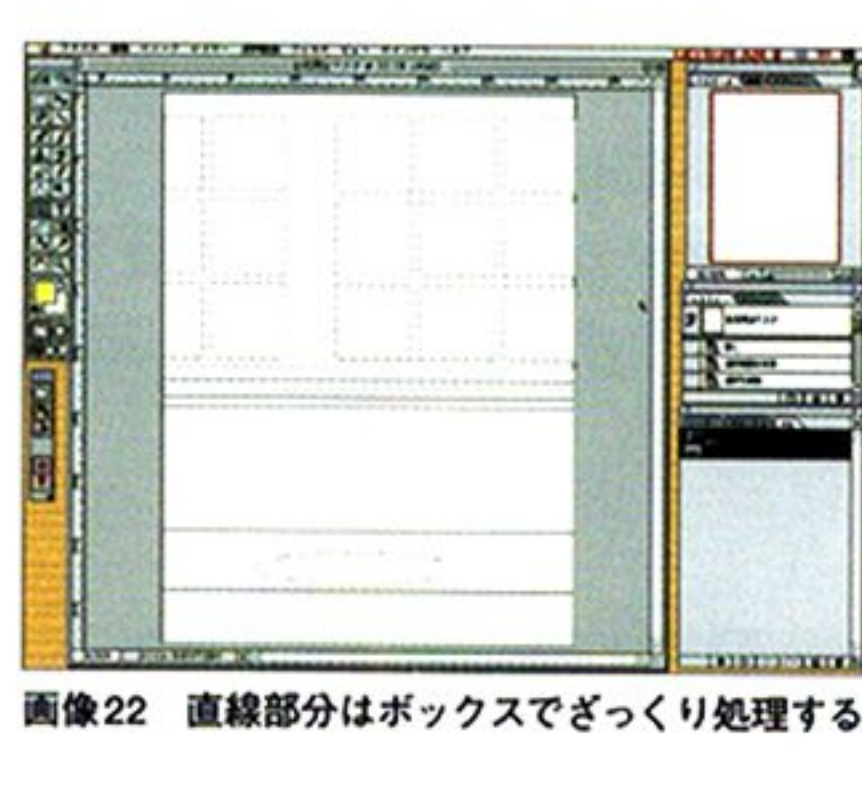
ぬいぐるみの部分では少し遊んでみました。本物のぬいぐるみをスキャンしてそのテクスチャを筆として新たなぬいぐるみを描画しようという試みです。まずぬいぐるみの素材を取り込みます。今回はペンギンを描くため,ペンギンに使用する色を意識して,取り込んだあとに色を調節しています(画像11)。これらの画像からぬいぐるみの質感をスタンプツールの「調整なし」でコピーして,人物の下絵に重ねるようにして描画します(画像12)。

画像11　実在の素材かを取り込んで色を調整

画像12　スタンプツールで縫いぐるみを塗っていく

画像13　ちょっとリアルになりすぎたペンギンの縫いぐるみ

画像14　ブラシを使って質感を抑え絵に馴染ませていく

画像15　今度は椅子を作っていく

画像16　アップにすると手描きだということがわかるだろう

画像17　背もたれの部分はちょっと注意が必要

画像18　あとで透けるようにパターン作成時にマスクを取っておく

画像19　いちばん後ろの背景を描く

画像20-A　消失点を決めて全体のアタリを取る

画像20-B　アタリだけ別レイヤーに移す

画像21　窓枠のマスクを作成する

画像22　直線部分はボックスでざっくり処理する

焼き込みツールなども使用して陰影を調節します。案の定，リアルになりすぎてしまいました(画像13)。ちょっとしたマスコットを描くには面白い手法ですが，これではどう考えても人物の絵に馴染みませんから，Painter上でエアブラシなどの筆を使って絵に馴染むように質感を抑え込みます。

標準添付のテクスチャライブラリの中からぬいぐるみの質感と似たような感じの紙のテクスチャを選んで拡大率を調整し，ブラシの手法にテクスチャを加えて薄目に塗り重ねていきます。修正後はだいぶ絵に馴染んできました(画像14)。

背景の制作

背景も基本的には手描きの技です。人物と違って直線ツールや楕円ツールなども多用しますが結局はデッサン力がものをいいます。この椅子などもかっちり描いてるように見えますが(画像15)，アップにすると手描きであることがよくわかります(画像16)。

コンピュータの手法として気を使ったところといえば，やはり背もたれの部分でしょう(画像17)。背もたれの網目部分は当時使っていたPainter3に標準でついていたテキスタイルだかパターンだかをそのまま塗りつぶしに使用したものですが，気をつけなければいけないのは，網目の向こうには背景が見えてなくてはなりませんから，パターンで塗りつぶした時点で同時にそのマスクもとっておかなければなりません(画像18)。網自身はあとからエアブラシなどで調子をつけることになりますので，あとになればなるほど自動選択ツールでも選択しにくくなります。

次に最背面のレイヤーになる床と窓です(画像19)。これはすべて直線で構成されているため比較的楽に描くことができました。といってもほかのレイヤーと矛盾がないようにパースはしっかりとらなければなりません。

まず基本となる窓や床の線を入れてから消失点を取ります(画像20-A ×の部分)。消失点を残したままだと描き進められませんからこれを別レイヤーに移します(画像20-B)。窓外はエアブラシを吹き付けてそれっぽく表現するだけなので，レイヤー分けする必要もないだろうと思い，窓枠のマスクを作ることにしました(画像21，22)。このように多少強引ですが直線部分はすべてボックス選択ツールで選択し，パスに変換しておきます(画像23)。これを元にして床と窓を描いていけば縦横の単純な線は楽に描けますし，マスクとしても利用できます。

壁を表現するには，テクスチャをうまく表に出すのがコツです。水彩薄筆で色をつけたあとに，そのまま白を選択してもう一度軽く上からなぞるとテクスチャの濃い部分が残ります(画像24)。下地の色とあまりにかけ離れた色を使うと気持ちの悪いまだら模様になるので気をつけなければなりません(画像25)。

木目も同じく水彩で表現しています(画像26)。水彩は最初から高い透明度を持っているうえに，別レイヤーに描画しているため，気に入らなけれ

ば下の絵に影響を与えずに消せるので使い勝手がよく，頻繁に利用しています。

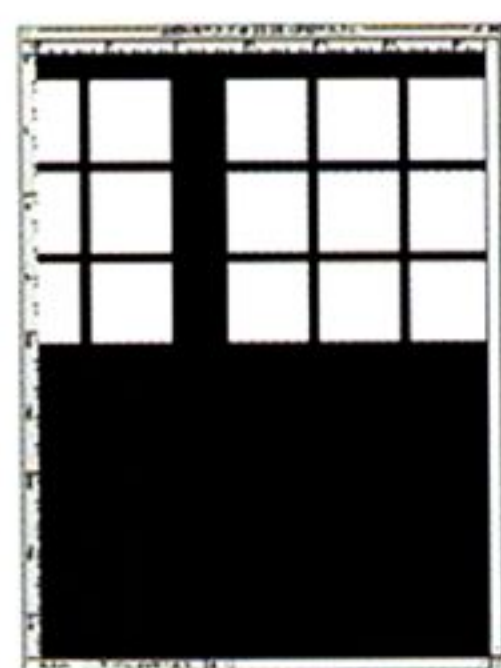
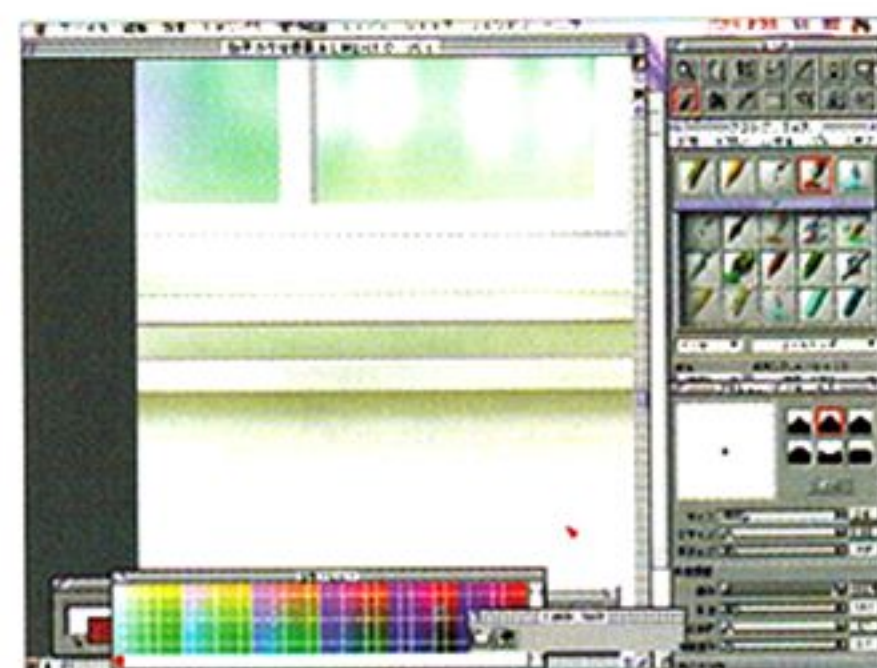
絨毯にはスーラタッチを使用してます（画像27）。とにかく今後のためにPainterの持ついろんな筆を試しておこうと考えていました。窓外は先ほどのマスクを利用して庭の木や垣根をエアブラシで表現し，光の感じを出してみました（画像28）。

最後はカーテンと観葉植物です。窓と床の線画状態を保存しておいてそれにあわせて描きます（画像29）。この頃はCG環境があまり整ってなくて，メモリも乏しかったので，全レイヤーを読み込んで描くという大胆なことはまだできませんでした。カーテンも植物も幾何学的な形状ではありませんから自由に筆を走らせられるのでついPainterのいろんな筆を試してみたくなります（画像30，31）。カーテンのレースの処理についてはあとからいくらでも描き直せるので，この段階で背景との整合性を気にすることはありません。

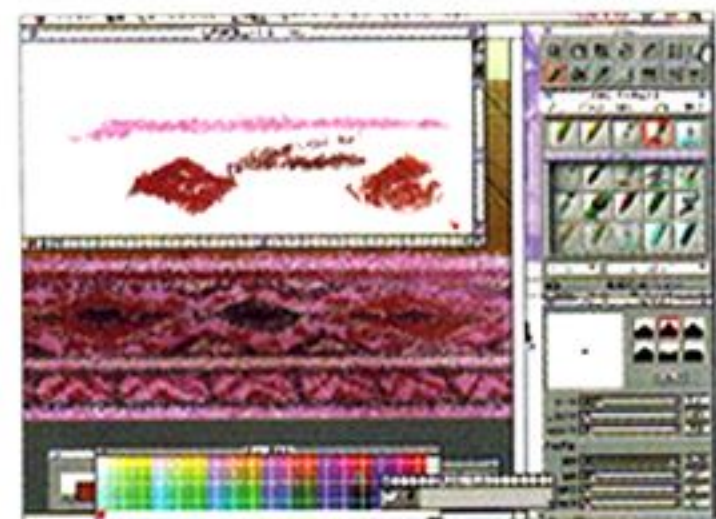
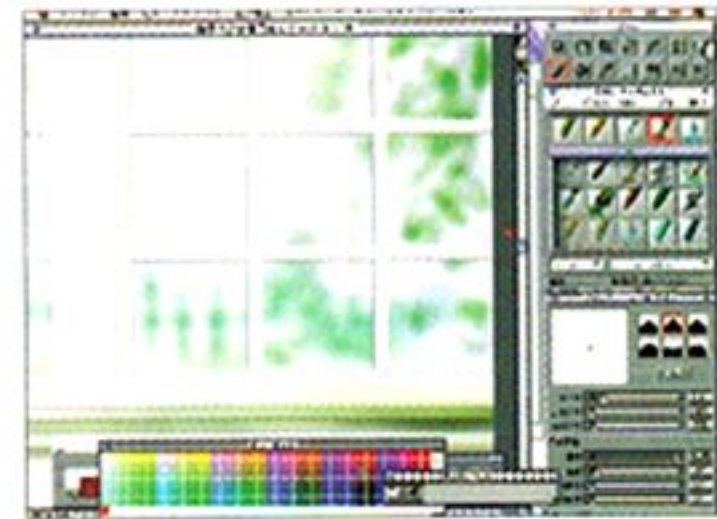
画像の合成

基本的にどのレイヤーも白地の上に描いているので，自動選択ツールを使えば，わずかの修正で合成用のマスクを作ることができます（画像32）。暫定的に合成してみます。別々に描いたせいもあって，色の整合性がとれていません。全体的にどぎつい色使いです（画像33）。窓外の明るさを強調するため，窓外以外の部分の明度・彩度を思い切って下げてみることにしました（画像34）。最背面のレイヤーはもともと窓外と一体になっていたので，マスクを利用して窓外の明るさを強調します（画像35）。

うまく色が落ち着いたら完成となるわけですが，実はここに至るまでにちょこまかと細部に修正を加えています。黒目を大きくしたりとか，帽子の色を変えたりとか，カーテンのレースを描き直したりとか……。

本来ならいちいちレイヤーに分けるような絵でもないのですが，この絵を描いたときはPhotoshop3.0のレイヤー機能を試してみたくて，わざわざレイヤーを使ったようなところがあります。

画像23　できあがったマスク

画像24　壁は水彩でテクスチャを生かすように

画像25　変なまだらが出てしまった

画像26　床の木目も水彩で描き込んでいく

画像27　スーラタッチを使った絨毯の表現

画像28　マスク越しに窓の外を描き込んでいく

画像29　最後にこいつらを加えてやる

画像30　Painterのすじ状ブラシで葉に表情をつける

画像31　カーテンも筆で自由に描き込んでいく

画像32　合成の準備でマスクを生成

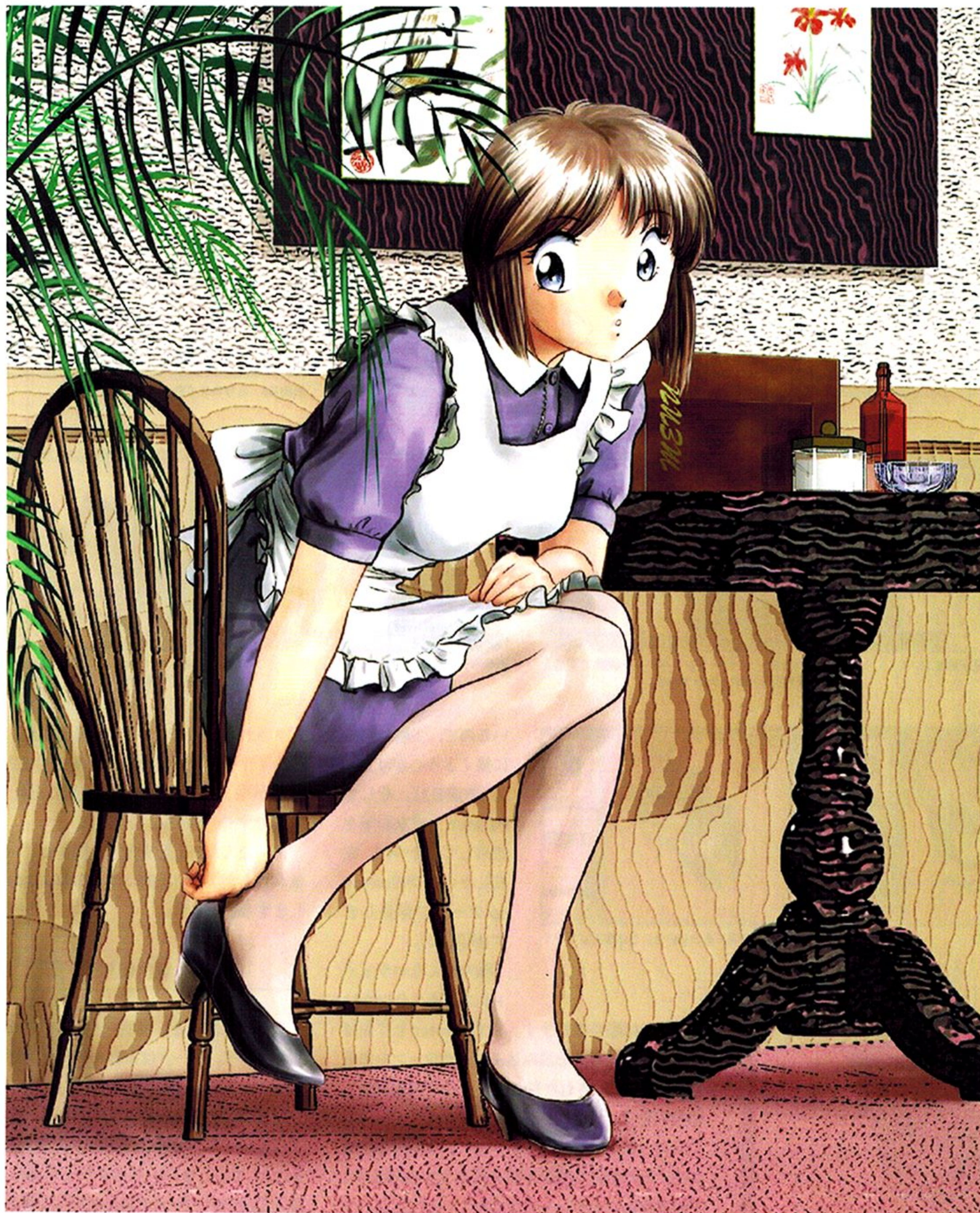
画像33　すべてのレイヤーを合成したところ

画像34　彩度を落として全体を落ち着かせてみた

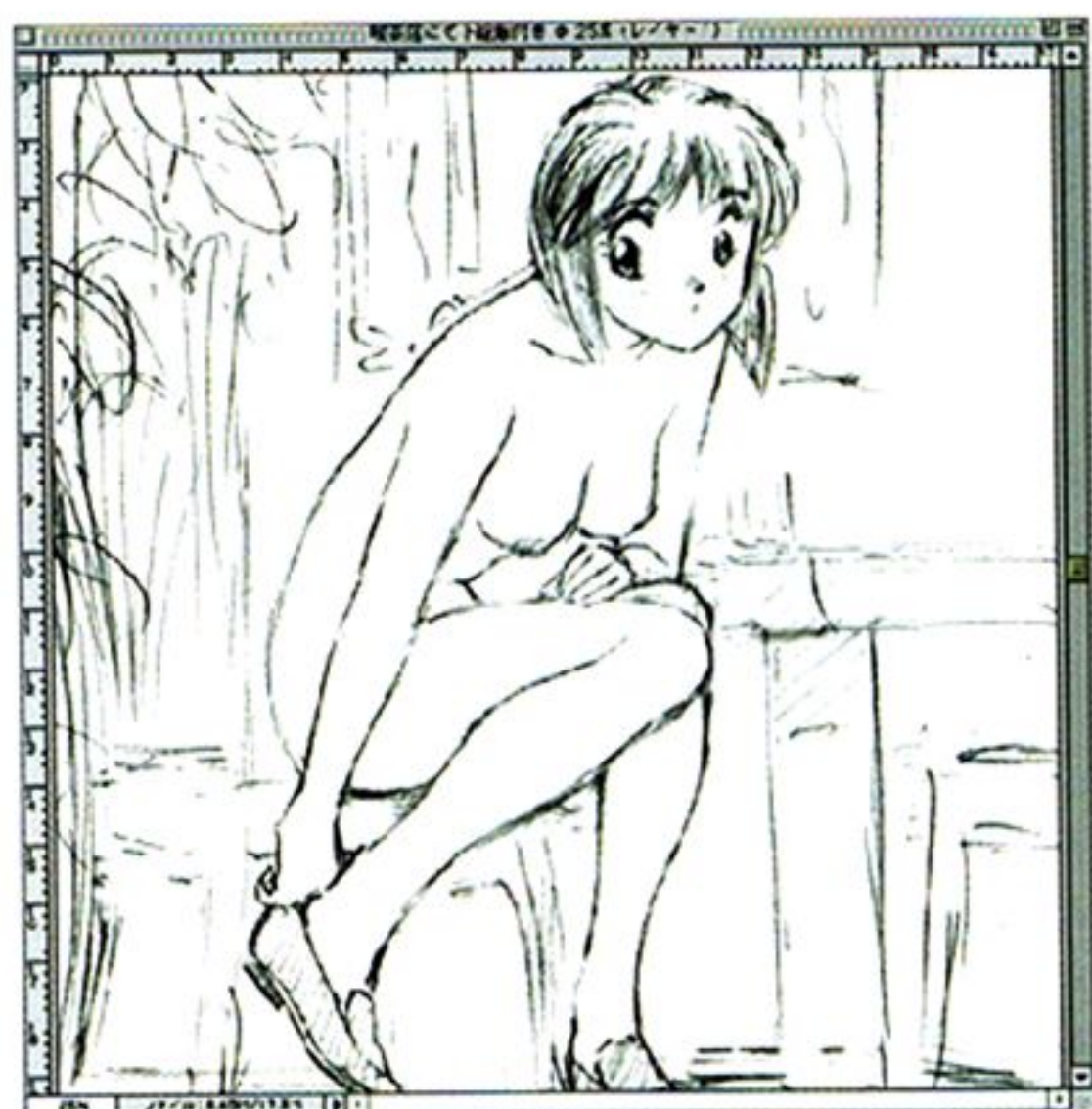
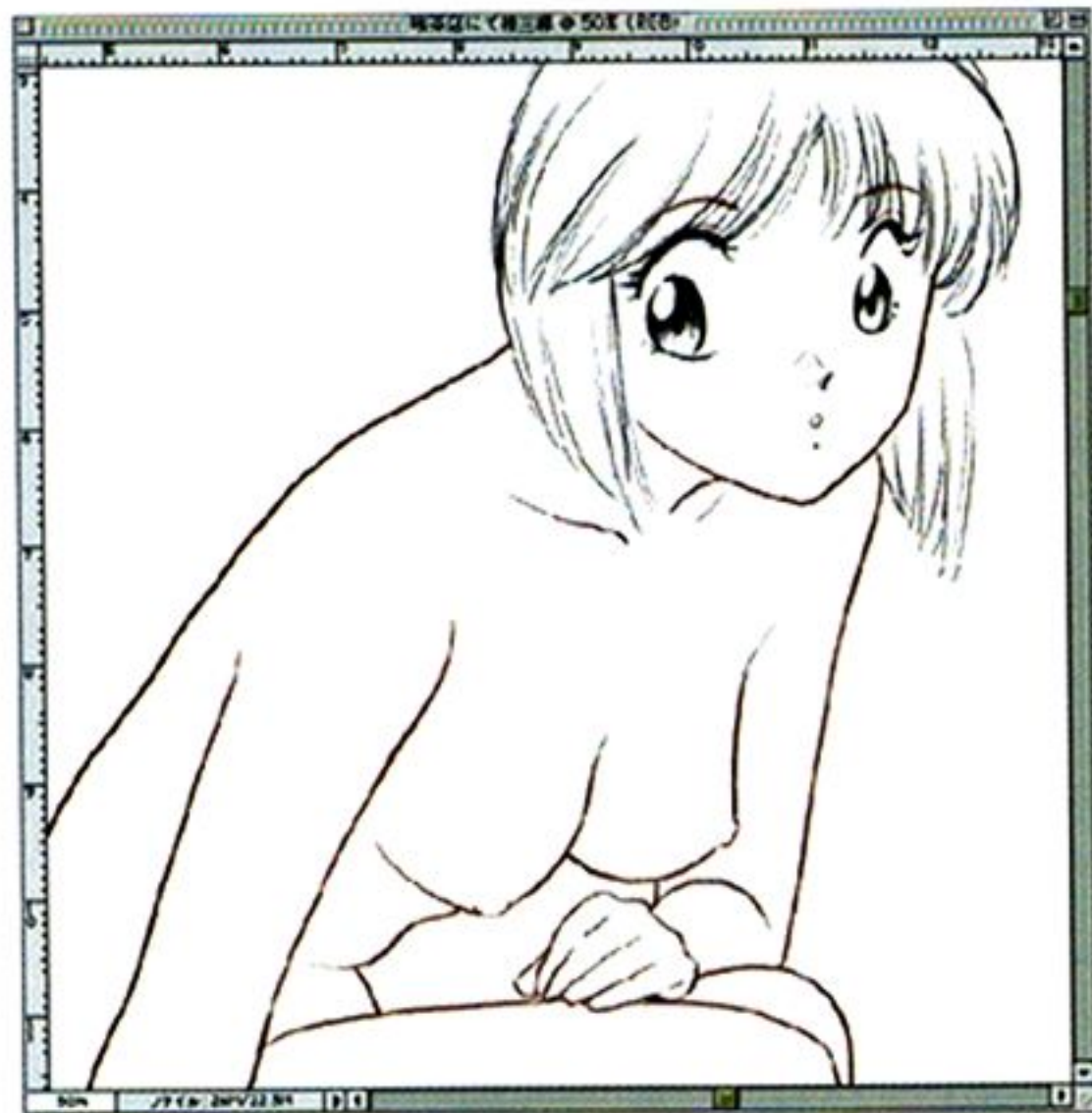
画像35　窓の外を明るくしてできあがり

Hisashi Morikawa

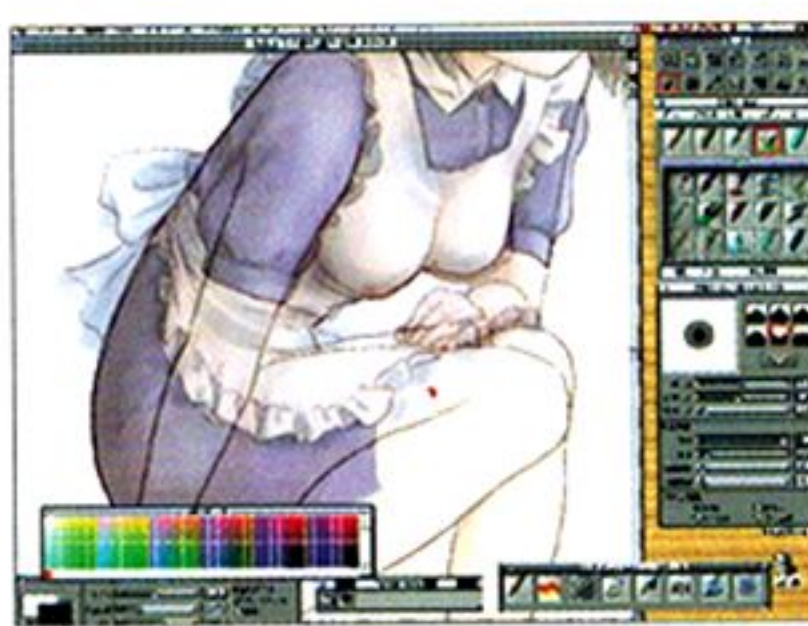
喫茶店にて

使用環境
PowerMac 7600/132 + BOOSTER 750/233
+ メモリ336MB
Caravelle PS-230DX2(MO) Seagate ST-3450/W(HD)
スキャナ EPSON ES-8000
タブレット WACOM ArtPad2pro

この絵のテーマは，以前喫茶店で見かけたウェイトレスさんの靴を直す仕草が妙にグッときたので，CGに起こしてみようと思ったのが始まりです。要するに見たまんまです。
「椅子の少女」が徹底的に手描きの技術にこだわったのに対して，今回はパソコンならではのアプローチで楽しく絵が描けないかということを追求しています。なお使用ツールはPhotoshop4.0, Painter5, ShadeProfessional R2, Expression 1.0.5などです。

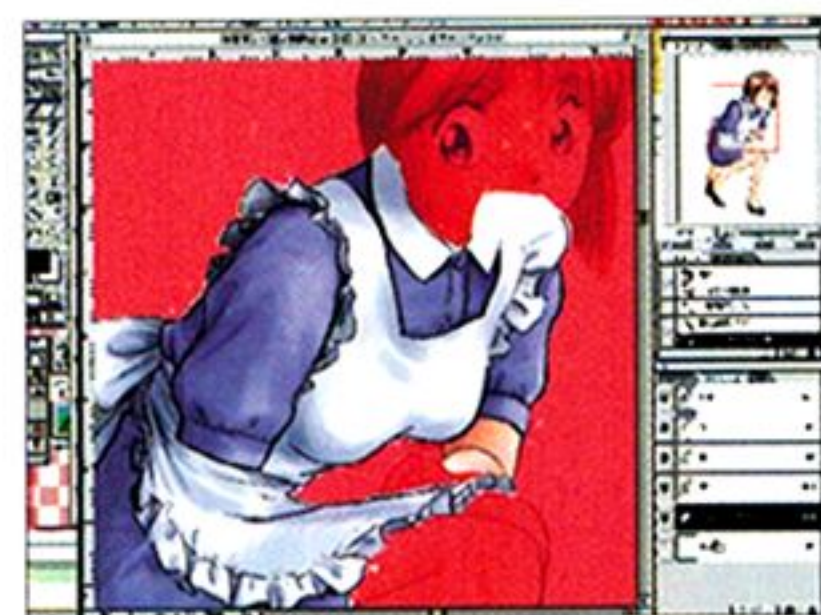
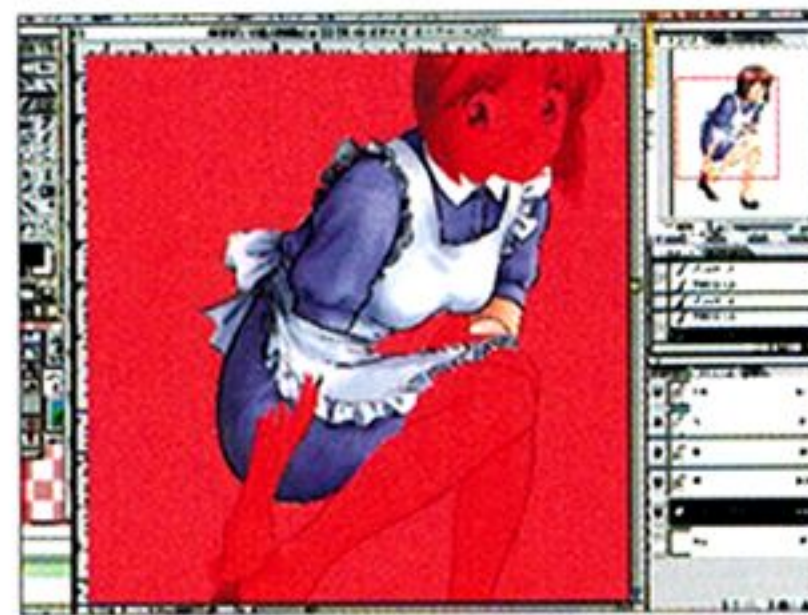
人物の制作

今回はウェイトレスの微妙な仕草を再現するのが目標なので，デッサンは重要です。したがって，下絵の鉛筆描きはヌードで行っています(画像B

画像B1　まず下描き。しっかりとポーズを決める

画像B2　キャラクターの下描きから主線をトレース

画像B3　とりあえず色を塗る

画像B4　人物の上に服を着せていく

画像B5　トレーシングペーパー機能で下描きを確認しながら塗っていく

画像B6　服の部分を仕上げる

画像B7　合成するために余白部分にレイヤーマスクを追加する

画像B8　合成していく

画像B9　うまくあわないところは修正していく

画像B10　合成完了。この手順は真似しないほうがいい気がする……

1)。ポーズがうまく決まったらスキャナで取り込んで「椅子の少女」同様の手法で下絵の線を薄くしてPainter上でペン入れをします。

今回はあくまでも漫画タッチでいこうと思っているので，主線は綺麗に描くよう心がけました(画像B2)。そしてPainter上で水彩筆を使用して色をつけ，完成したら再びPhotoshopデータで保存します(画像B3)。

服を着てないウェイトレスはあまりいないので，次は服を描きます。まずPainter上に先ほどの人物完成画像を読み込んで，クローンを作成します。コピーされた絵そのものは必要ないので削除します。

トレーシングペーパーをチェックして人物を透かせ，それを元に服の下絵を描きます(画像B4)。下絵を描いたらトレースは必要ないのでトレーシングペーパーはOFFにしておきます。しかし描いてるうちに下絵からずれることはよくあるので，ときどきトレーシングペーパーをONにして確認しながら作業するのがよいでしょう(画像B5)。今回は服の着色も人物と同じく水彩を使用しています。完成したらこれもPhotoshopデータで保存します(画像B6)。

完成した服と人物をPhotoshop上で合成します。服を人物の上のレイヤーに配置し，自動選択ツールで白地の部分を選択して(画像B7)「レイヤー」→「レイヤーマスクを追加」→「選択範囲をマスク」を実行すると服の周りの白が大ざっぱにマスクされた状態になります(画像B8)。あとはこのレイヤーマスクをひたすらちまちまと修正し，人物が服より上になる部分は削り出していけば完成です(画像B9)。

……なんだか手順を大きく間違えているような気がしますが，結果的にできあがったのでよしとします(画像B10)。

背景の作成

背景には試みとして3DソフトのShadeを使用しています。データは市販のモデリングデータ集「3DPOCKET」から持ってきています。

まず下絵を参考にテーブルや椅子，後ろの壁などを配置して，カメラアングルを決めます(画像B11)。このときパースペクティブ画面に下絵が読み込めればパースをあわせるのに都合がよいのですが，残念ながらその機能は次期バージョンのR3を待たなくてはなりません。ここでは勘でパースをあわせます。下絵の感じからテーブル面のわずかに下あたりを真横から見るようにカメラをセッティングするとちょうどいいようです。各オブジェクトの大きさの比率は実際にその空間に人間が立っていてもおかしくないかどうかを基準にチェックすると，あとで人物と合成するときでも違和感なく合成できます。もちろん，それでも人物とぴったりあうことは稀ですから，レンダリングしたあとにPhotoshopなどで修正することになります。

テーブルや椅子には木のテクスチャ，壁にはでこぼこのテクスチャを貼ってライトの方向を調節し，レンダリングします（画像B12）。高解像度の絵に馴染ませるにはそれなりの大きさでレンダリングする必要があります。ここではマシンスペックと相談して2000×1500ピクセルのサイズでレンダリングしています。ちなみに実際に描いてる絵はこの倍ぐらいあります。

さて，レンダリングしたのはいいのですが，このままではリアルすぎて漫画の絵に馴染みません（画像B13）。そこでPhotoshop上で「エッジのポスタリゼーション」をかけてみました（画像B14）。最適な線の太さや強さは絵の大きさによって変わるので一概にどの設定がいいとは限りません。いろいろ試してみるといいと思います。

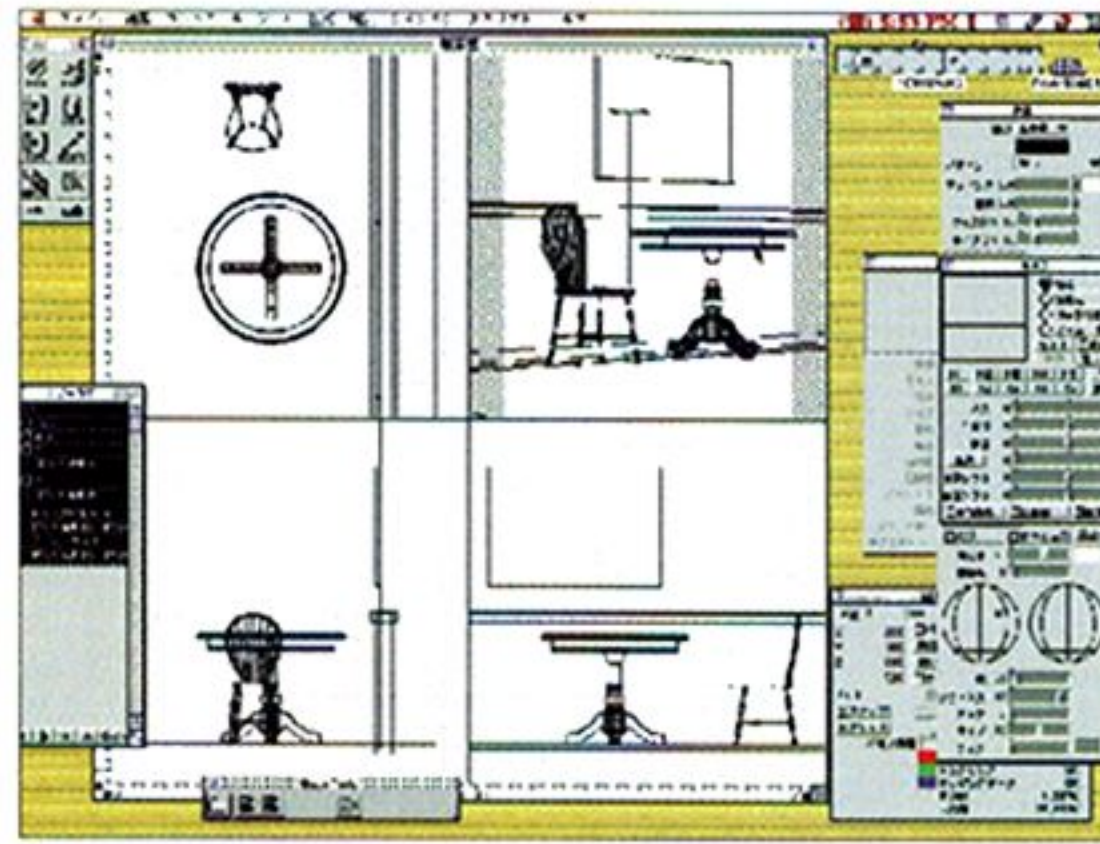
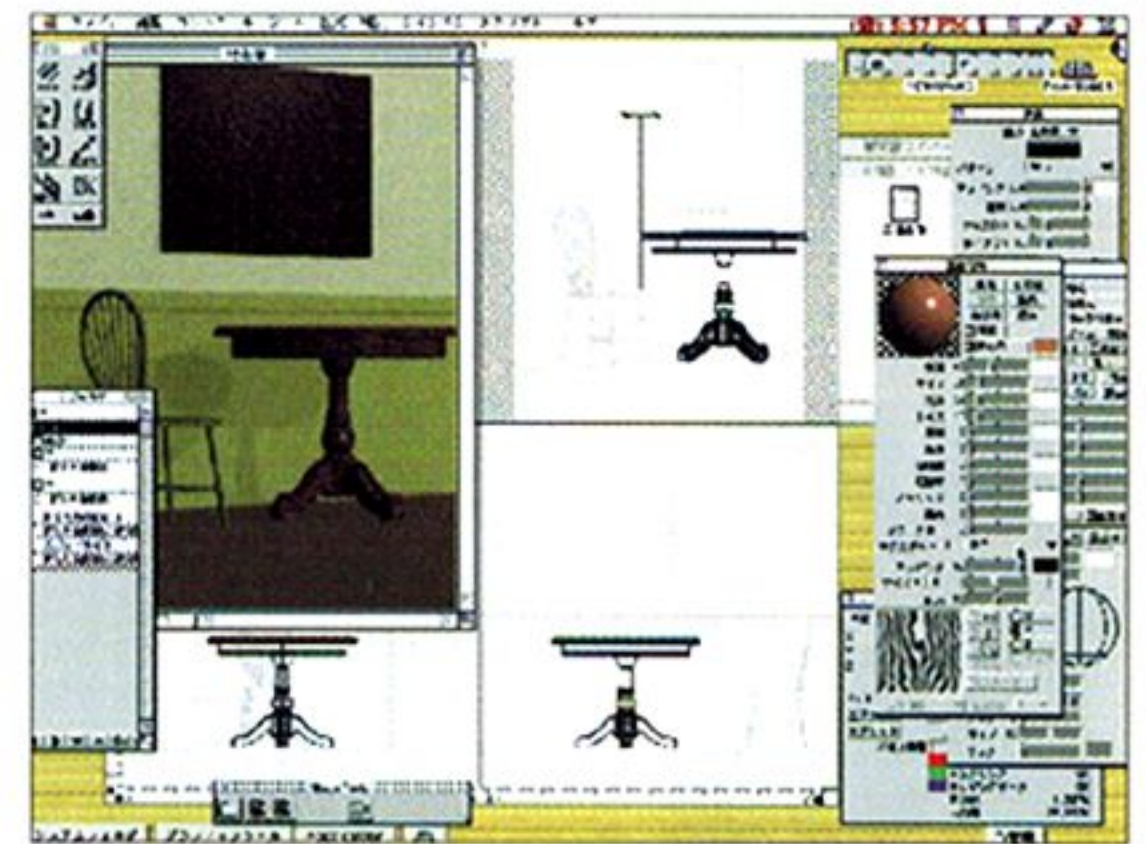
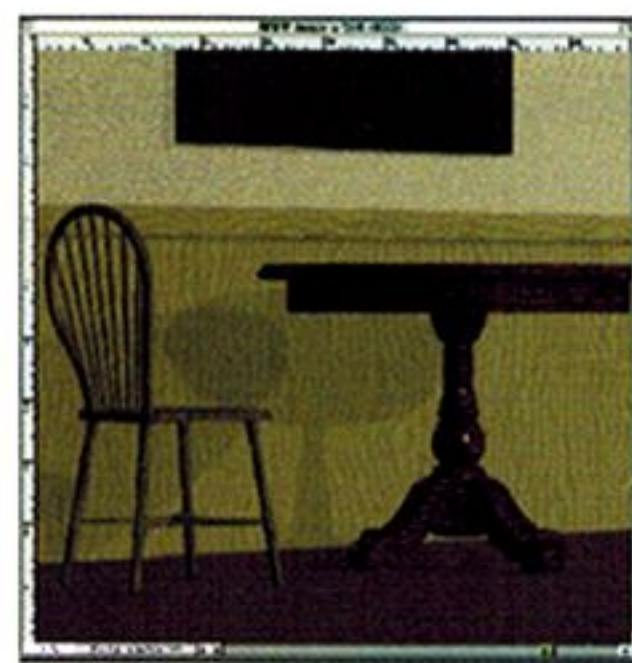
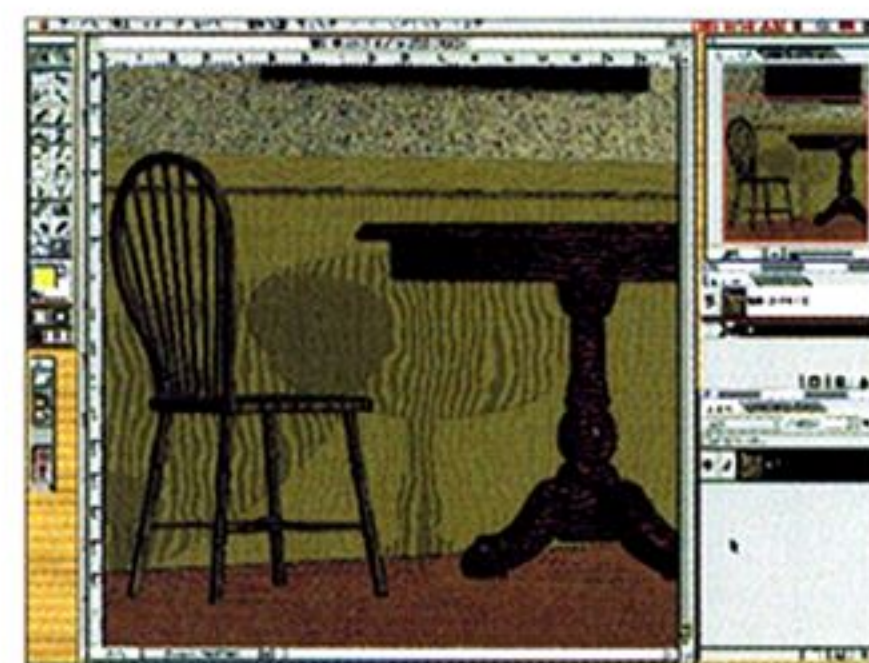
テーブルの上の小道具など複雑なオブジェクトが増えると3Dの処理が重くなるので，テーブルや椅子などの背景とは別々にレンダリングします（画像B15）。ここでもやはりパースペクティブ画面に下絵を取り込む機能がほしいところですが，ないものはしょうがないのでこれも勘でパースをあわせました。これもバックのテーブル，椅子にあわせて「エッジのポスタリゼーション」をかけます（画像B16）。

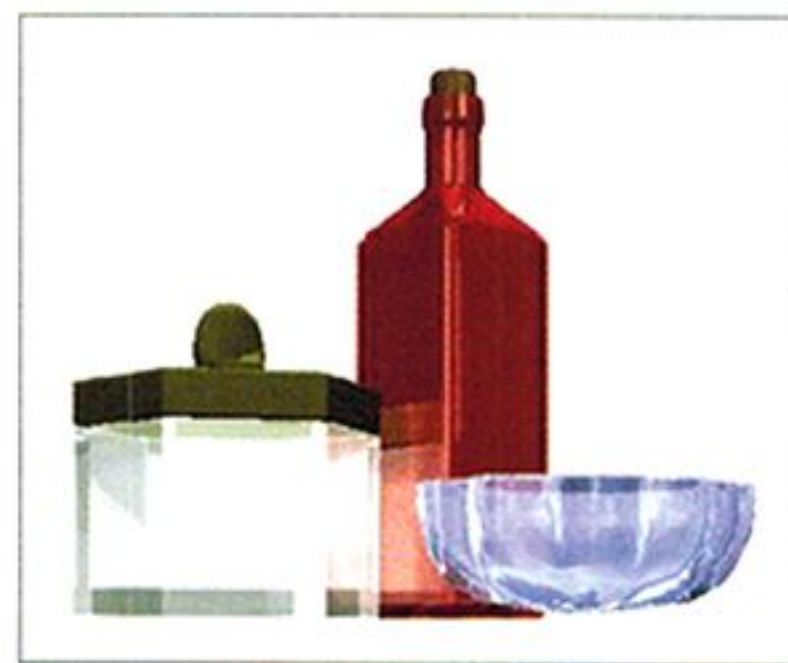
メニューはほとんど平面なので，わざわざ3Dソフトを使うまでもありません。四角を描いて直線で厚みをつけ，模様は雲模様，文字はその選択範囲を利用してグラデーションをかけているだけという実に安直な作りです（画像B17）。あとはパースに沿って変形するだけです。後ろの絵も同様で，以前に「Mac書道Pro」の練習で作っておいた絵を単に貼り付けただけです（画像B18）。

残りは手前の観葉植物だけですが，そのまま手で描いたのでは面白くないので，私のお気に入りのExpressionを使ってみました。この手の植物ならあっという間に描けてしまいます。まず笹の葉のような形をした既存のストロークで植物の葉を1本のストロークとして定義します（画像B19）。定義した1本のストロークからさまざまな形の葉が描けるのはなんともいえない楽しさがあります（画像B20）。できあがった植物はEPS形式で保存してPhotoshopデータにコンバートします。

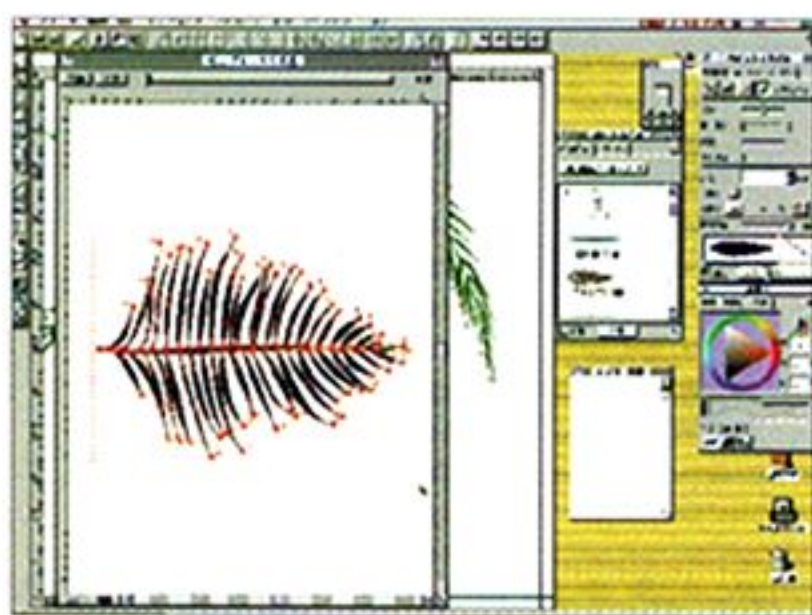
Photoshopのペンツールで葉に質感を与えるとそれらしくなります（画像B21）。「透明部分の保護」をチェックしておくとやりやすいでしょう。

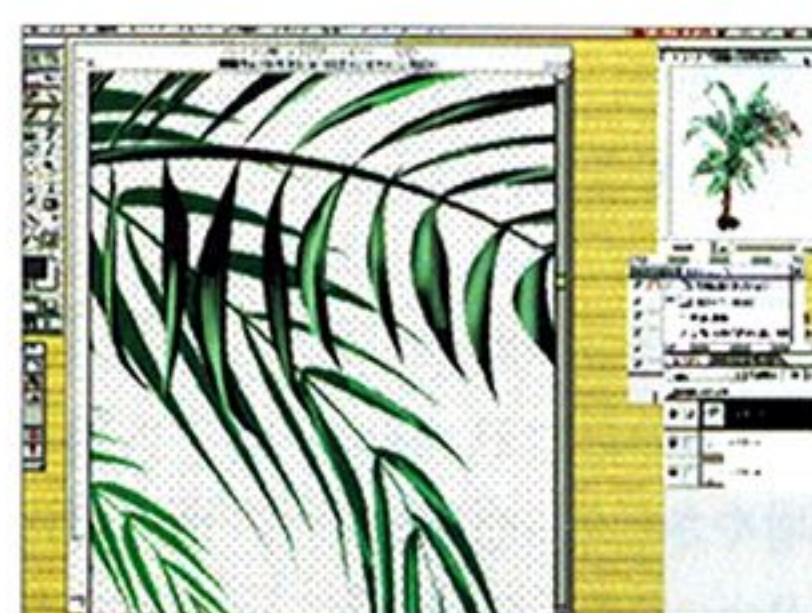
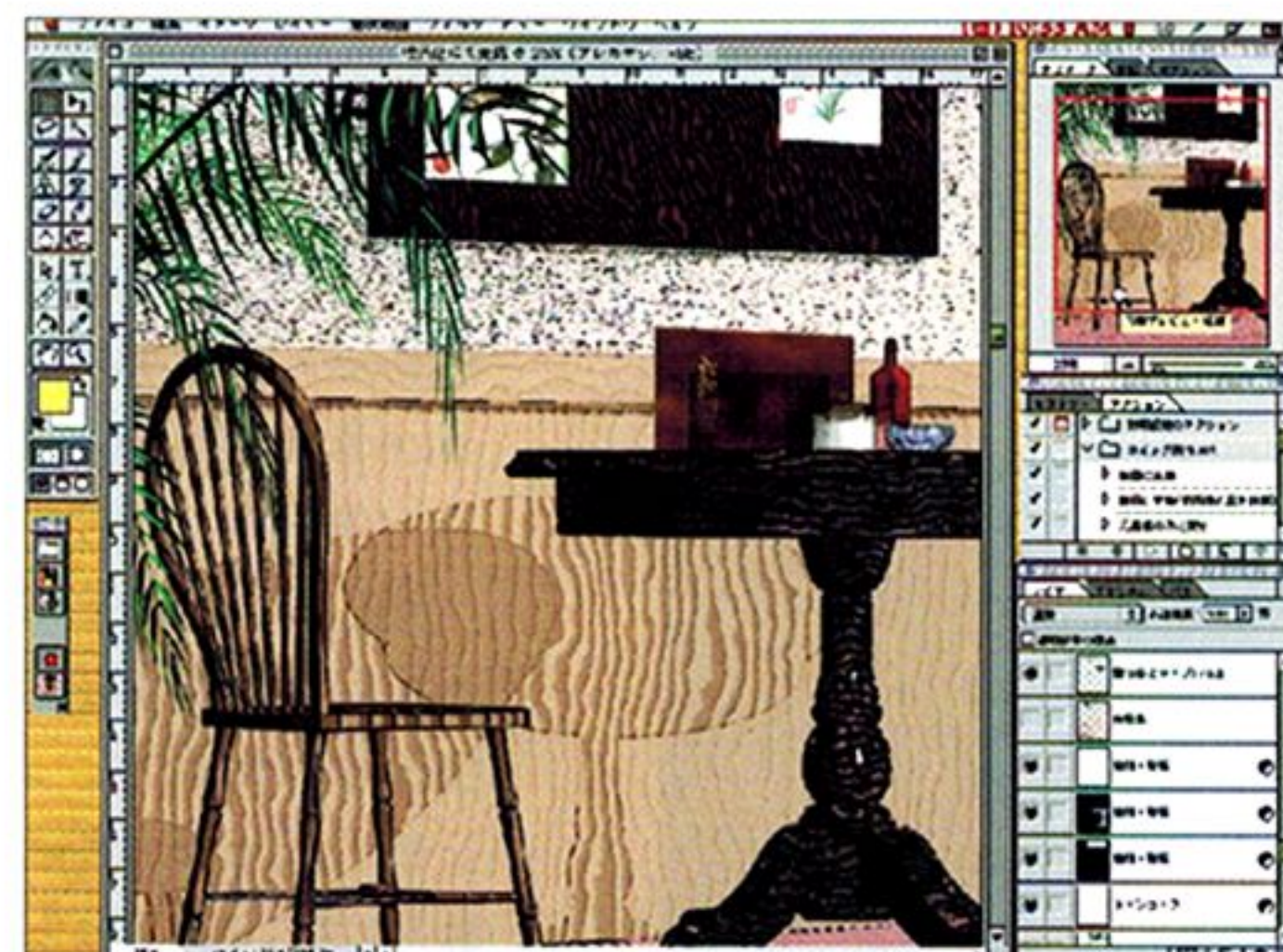
画像の統合

できあがった各素材を統合します。背景は全体のバランスを考えながら色調補正します（画像B22）。人物が違和感なく収まったら完成です。

画像B11　3Dツールのオブジェクトを配置して背景を作っていく

画像B12　ライティングなどを調整してとりあえずレンダリング

画像B13　レンダリング結果

画像B14　ちょっと絵柄とあわないので，フィルタをかけて馴染ませる

画像B17　メニューを描く。手描きでもこれくらいは大丈夫

画像B15　小物は別で設定してレンダリングしておく

画像B16　これも絵柄にあわないので，フィルタをかけた

画像B18　壁の絵はMac書道の練習作だった

画像B19　観葉植物は専用ソフトを使って生成してみた

画像B20　B19のパターンからこういうのが生成される

画像B21　葉っぱにペンツールで質感を加えていく

画像B22　各パーツを合成して，全体的に色調整をしていく

野澤絵美

Emi Nozawa

天使が舞い降りた夜

天使が夜にそっと泉に降りてくるグラフィックです。前々から作ってみようと思ってはいたけど，大変そうで尻ごみしていました。今回は雑誌用にとのことで挑戦して作ってみました。

私が初めて覚えたソフトウェアがSTRATA STUDIO PROです。3Dグラフィックを覚えようとしてCG専門学校のデジタルハリウッドへ行ったときに，学校で教えていたのがこのソフトでした。課題の3DCGを作るのに自宅にもソフトが必要で購入。そのまま，ずーっと浮気もしないで，これ1本でいろいろな3DCGを作っています。仕事ではインターコム製品（まいと～くFAX V.3など）のソフトウェアに使用しているスプラッシュグラフィックが代表作です。会社のMacにはメモリが32MBしかないのですが，意外と快適に作成可能です。ただしそれもバージョンが1.75のときの話で，去年発売になった2.0以降はメモリを湯水のごとく要求するようになり，自宅のマシン（メモリ176MB）でも太刀打ちできなかったので，1.75を使い続けています。

現在，STRATA STUDIO Pro 2.5へのバージョンアップのお知らせがきているのだけど，Mac OSが8以上だし～，メモリ増やさなきゃならないしで，どうせならじーさん（PowerMac G3系）を購入して，そっちに入れたほうがいいのでは……と悩んでいます。

3Dソフトといえるかわからないのですが，むちゃくちゃ重宝しているがKPT Bryceです。これは景色を作成する専門のソフトで，海，空，山といった背景を簡単に作り出してくれます。素材集を揃えるよりKPT Bryce1本あったほうが自分の希望する背景が手に入るので助かっています。ちょっとバグが多いのですが難点でしょうか。1995年代のThe Windowsも結構これで作成したCGグラフィックで表紙を飾っていました。あの当時は綺麗だな～と思っていたが，実はこれだったのかぁ～と懐かしく思い出しています。このCGにも背景の空を作るのに使用しました。

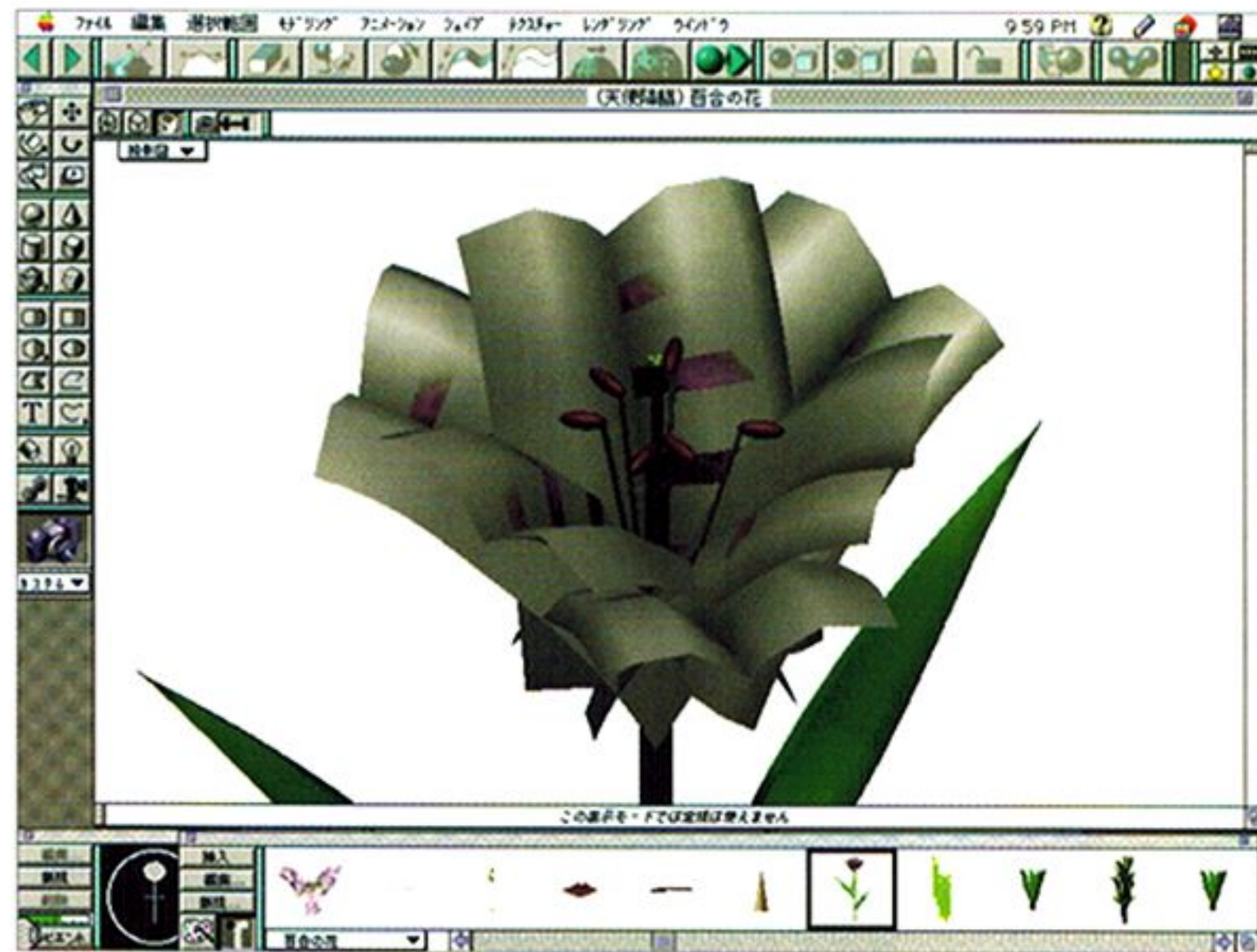
メインCG

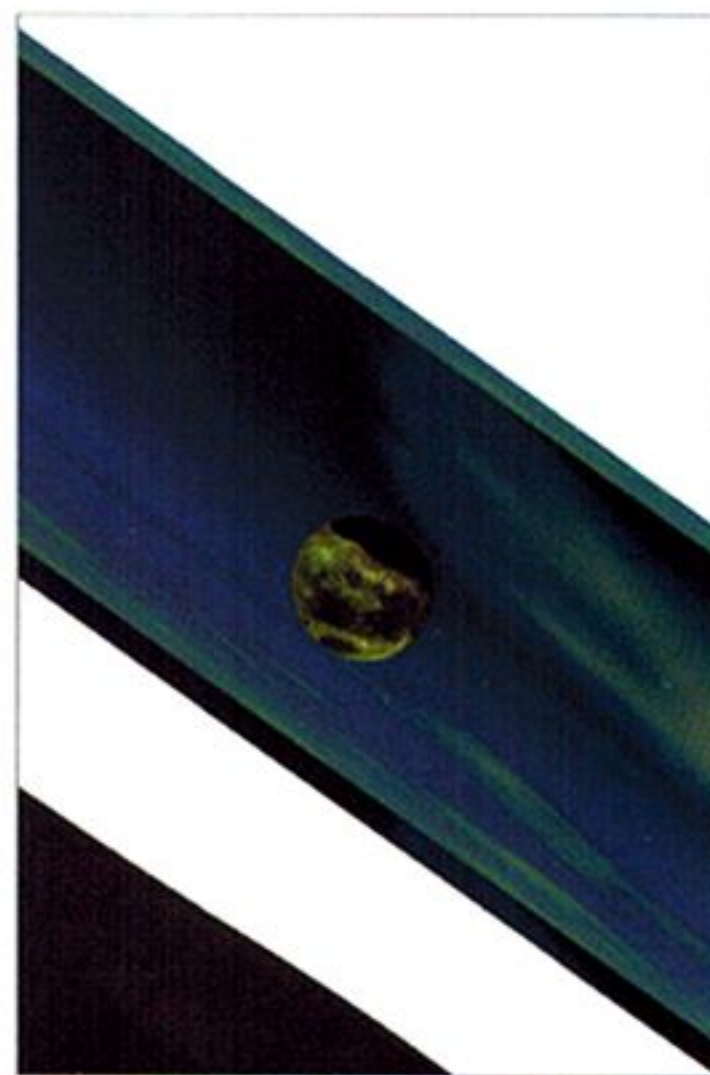
天使が泉に舞い降りたところ。場所はバリ島あたりをイメージに作成。羽とかもう少し薄くしたかったけど，それは次回の課題にしましょう。

天使全身

人物のモデリングってへたっぴなんですが，一所懸命作りました。1.75はボーンスケルトンやベジエ曲線といった機能がなく，曲線のモデリングは結構たいへーんなのです。服の曲線部分，腕，

使用環境
ソフトウェア：STRATA STUDIO PRO 1.75
：KPT Bryce
マシン：Macintosh7500/100
メモリ：176MB
作成時間：約50時間

とにかくパーツ数が多くて大変そう……だけど，羽はむしろ基本パーツの組み合わせで大丈夫。問題は曲面部分ですね

このような曲面に花びらをマッピングしていくと，かなり複雑な形状になる。基本構造をしっかりと押さえよう

顔は近くで見るとこんなにシンプルだけど，全景で見ればわからない。マッピングで勝負か

しかし，Bryce使って空だけ描きますか，この人は……

背景空間はこんな感じ。キャラクターを中心に構成していく舞台装置のようなものですね

髪の毛，帯といった曲線の部分はすべて多角形を組み合わせ，少しずつ角の位置をずらしながら形を作る方法をとったら，時間がえらくかかったというか，作業時間のほとんどがこの天使さま作りに取られてしまいました。

天使顔アップ

顔はテクスチャ張り付けで対処。時間があればむちゃくちゃ書き込みたいけどアップにしない作品だったので，目鼻がわかればいいや〜ってレベルにしておきました。実は，STRATAでの顔のモデリングはかなり難しいのです。同じ3Dソフトであるshadeってソフトのほうが楽なので，昔，Shadeで作成してSTRATAに変換してみたこともありましたけど，変換が全然へぼい感じで挫折。友達に聞いたら，友達も同じ失敗をしたとのことで，STRATAユーザーが一度は通過する儀式なのかな……と思い，それ以来，根性でSTRATAで作成するようにしています。

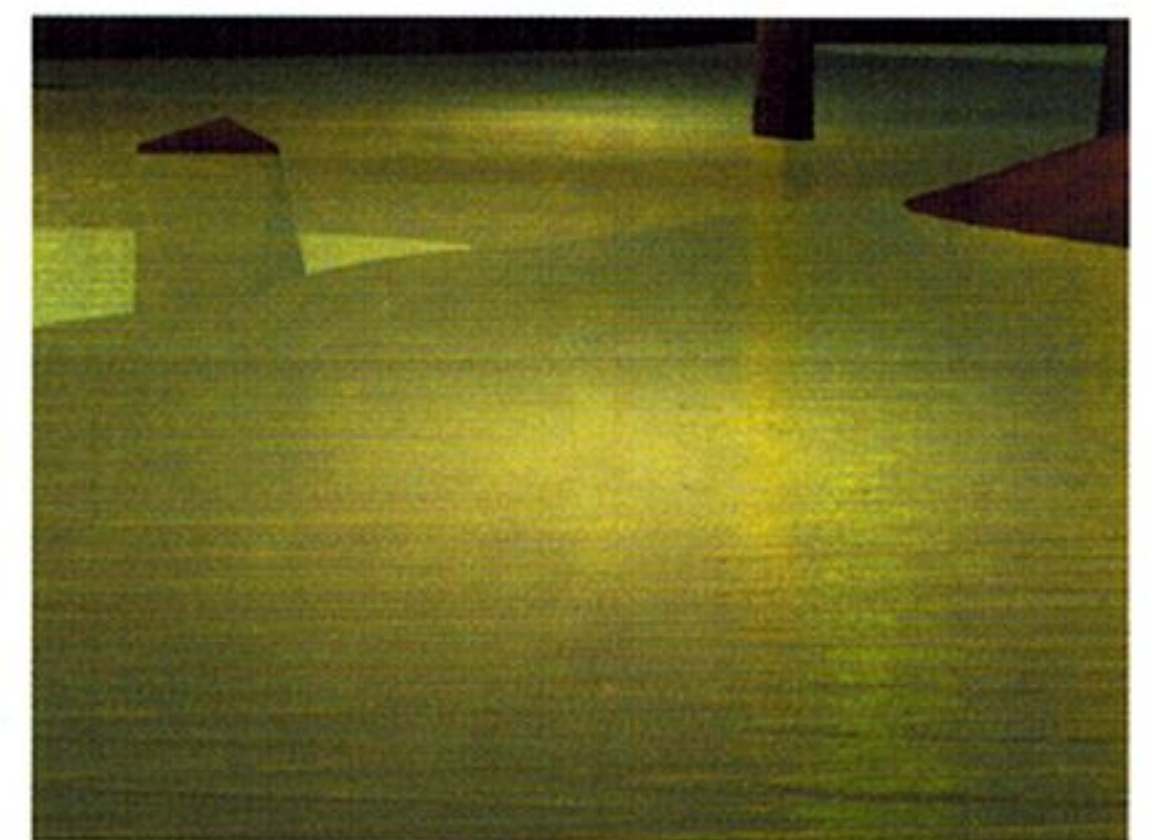
ユリ

ユリの花は前面に出したかったのでテクスチャをきちんと作成。実際の花をスキャナで取り込んで加工したのを，花の形に曲げた板に張り付けて一丁あがり！ このくらいのグラフィックだと1時間もあれば楽勝です。花の作成方法は覚えておくとあちこちで使えて重宝しています。

空

1枚の板にKPT Bryceで作成した空を張り付けます。そして球体に月のテクスチャを張り込んだお月さまをぽっかり浮かべました。舞台装置の要領。肝心なのは板と月は影をナシの設定にすること。じゃないと，月の影と空の影が登場しちゃってとても変な世界になります。

全景

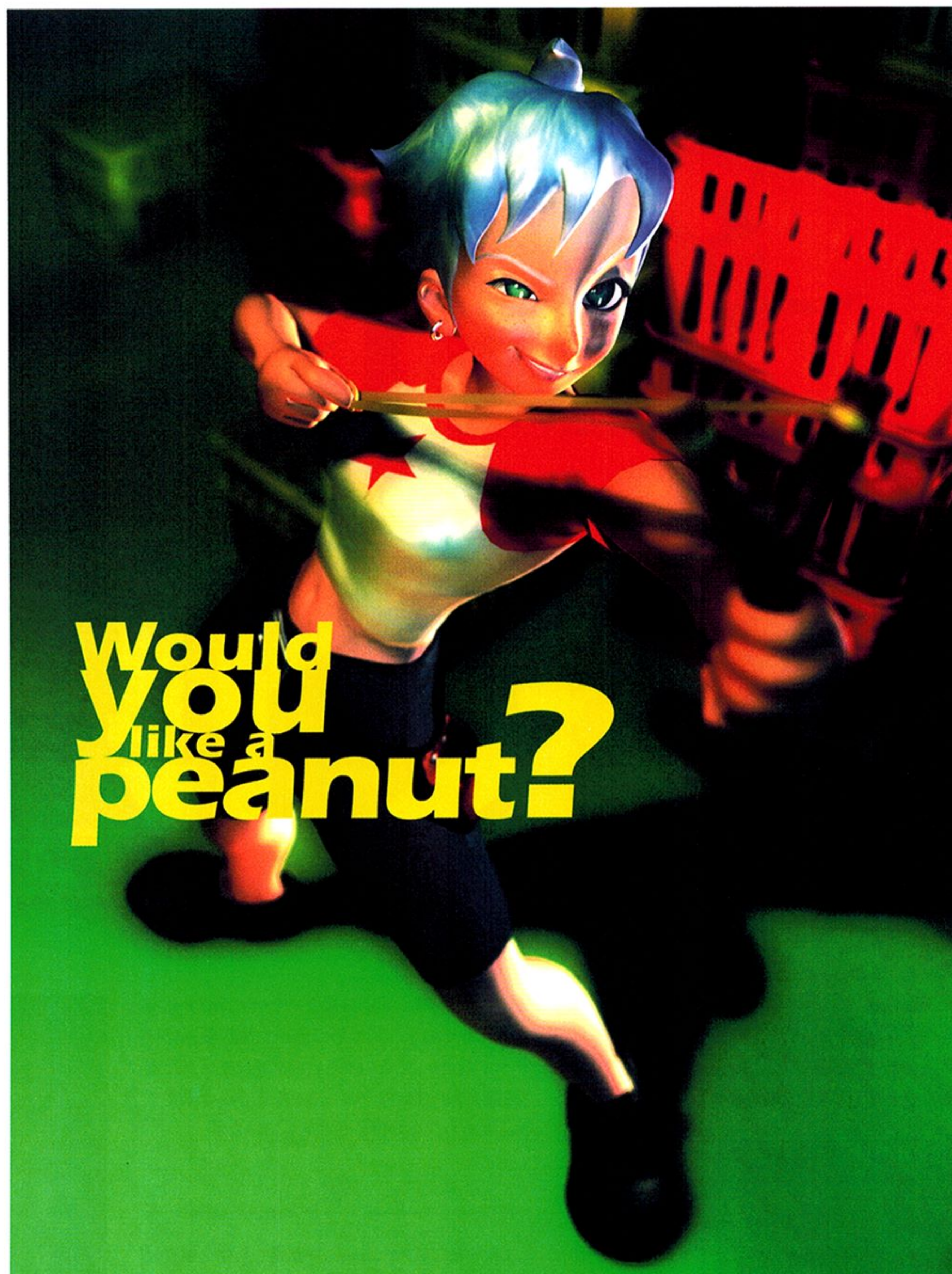
昼間の光量で全景を撮影したもの。見えない部分をテキトーに作っているのがバレバレだけど，これでいいのだ。下中央の四角いのがカメラ。ここから見えないものはとりあえず省略しちゃう。じゃないと，レンダリングによけいな時間がかかっちゃうから。

バンプマッピングによる水面画像。バンプマッピングの主要な用途かも

泉

3Dソフトによっては水の波紋を表現する機能もあるのですが，STRATA1.75にはそんなありがたいものはない。なので，Photoshopでテクスチャを作ってバンプ機能で水の波紋を表現。結構なんとかなる（編注：それが普通のやり方です）。……なんとかなるけどそれに気づくまでに多大な努力と多大な失敗が累々たる死体となり横たわっています。3Dは根性根性ど根性で作成するしかありません。

由水桂

Kei Yoshimizu

Would you like a peanut?

ピーナツはいかが？

今回は，リアルすぎない，クレイのようなアニメのようなノリで，悪ガキっぽい元気な(でもちょっぴり可愛いような)イメージで作ってみました。

使用環境
- 3000×4000pixel（レンダリング時間約5時間）
- Windows95 PentiumⅡ 300MHz 192MB Memory
- Lightwave3D 5.5 / Photoshop / Painter

復刊おめでとうございます

はじめまして，由水と申します。私も生っ粋のX68000ユーザーのひとりであり，当然熱烈な読者だったので，こうしてここで駄文を書かせていただけるのは非常に喜ばしい限りです。そもそも私が3Dを始めたきっかけがOh!X 付録のDoGA CGA Systemでした。思い起こせば6年前（！）CAD.X で一所懸命顔のモデリングをしていた頃が思い出されます……。

と，初めての読者が引いちゃいそうな昔話はおいといて，さっそく解説にまいりましょう。

Lightwave3D

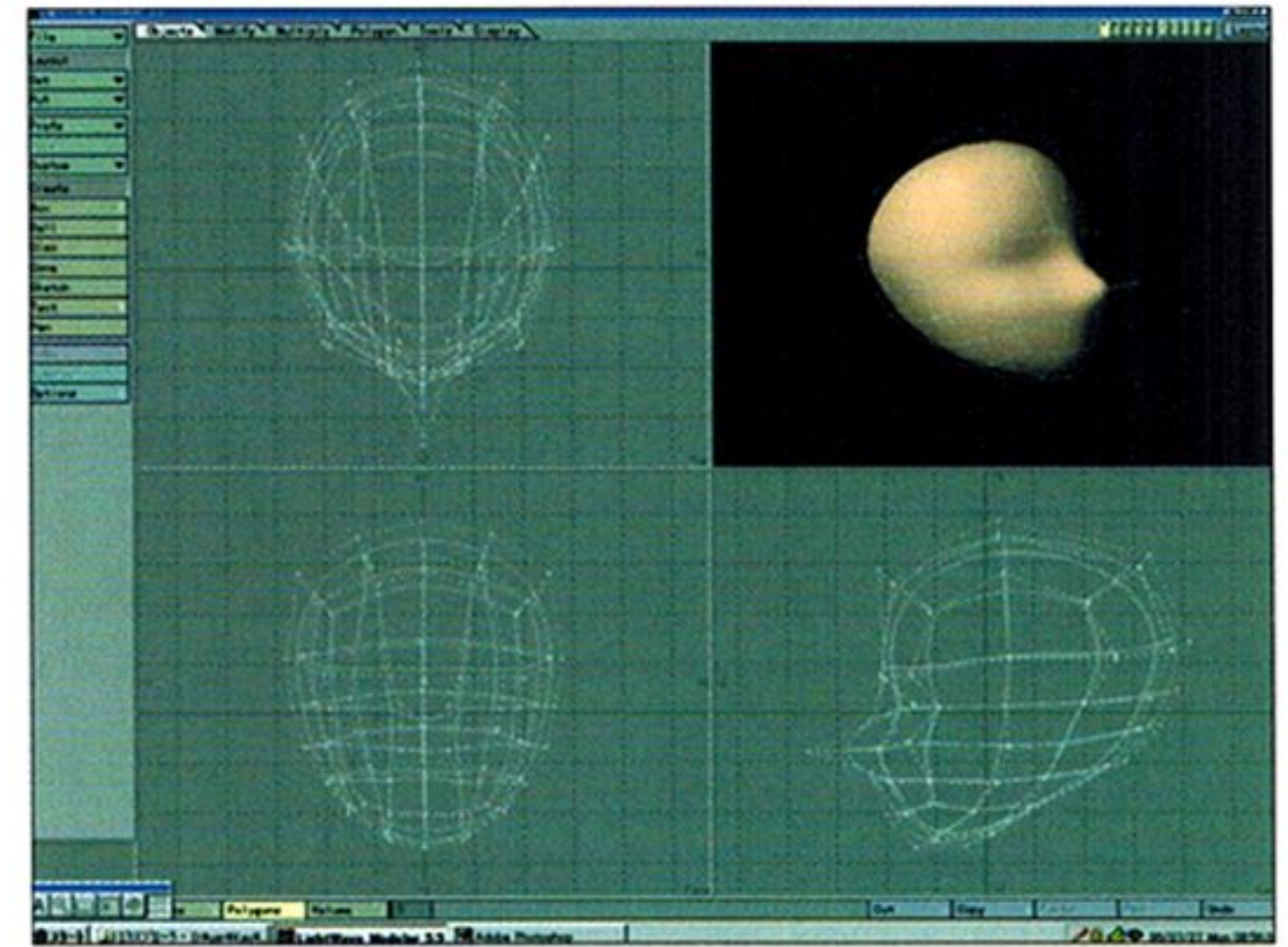
最近では妙に勢いのいいソフトなので，3Dの経験がない方でも一度はその名を耳にしたことがあるんじゃないでしょうか。Lightwave3D（以下LW）は私がDoGAの次に出合った3Dソフトです。その頃はアミーガで一所懸命……とかそんな話は止めときましょう。

さて，LWの最近のバージョンではモデラにMetaNURBsと呼ばれる機能がついていて，有機的な造形をまるで粘土を扱うように造ることができます。私はどちらかというとメカや建物よりもキャラクターなど有機的な曲線を持ったモデルを制作することが多いので，MetaNURBsは手放しがたい機能のひとつです。

モデリング

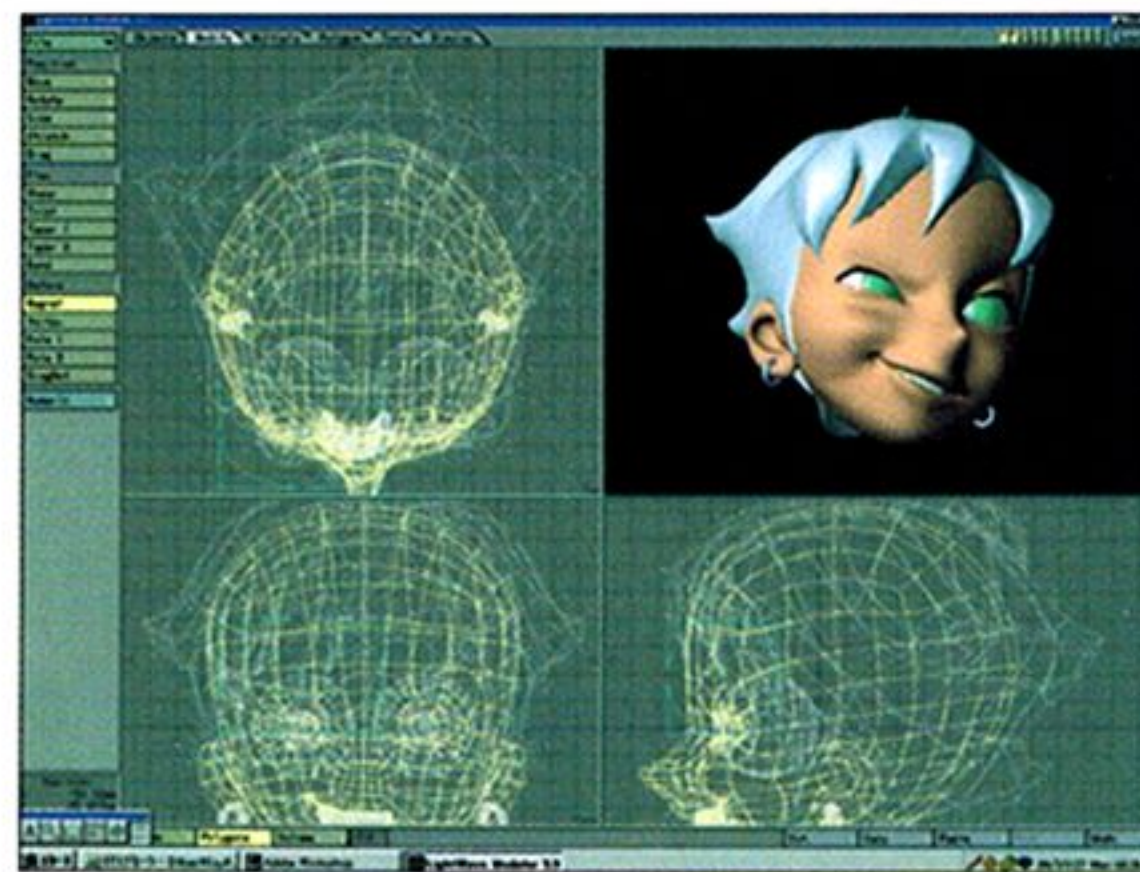
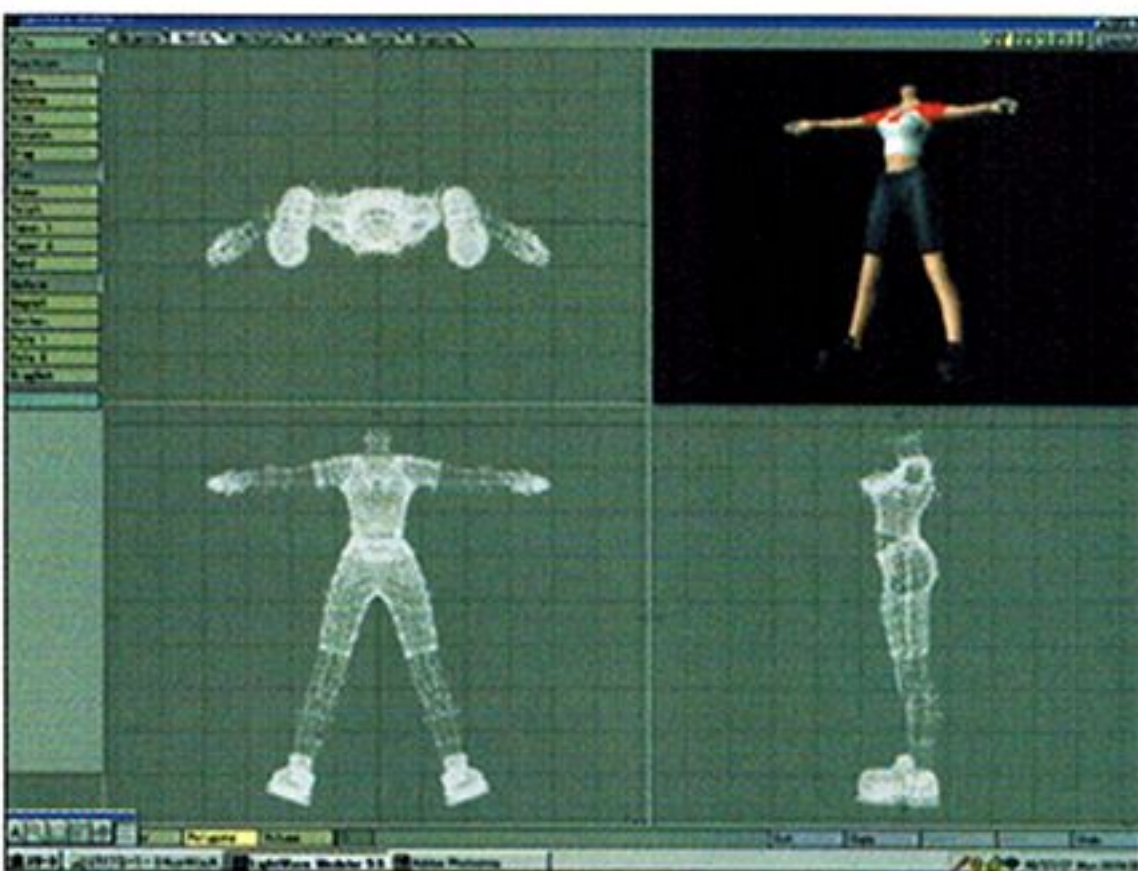
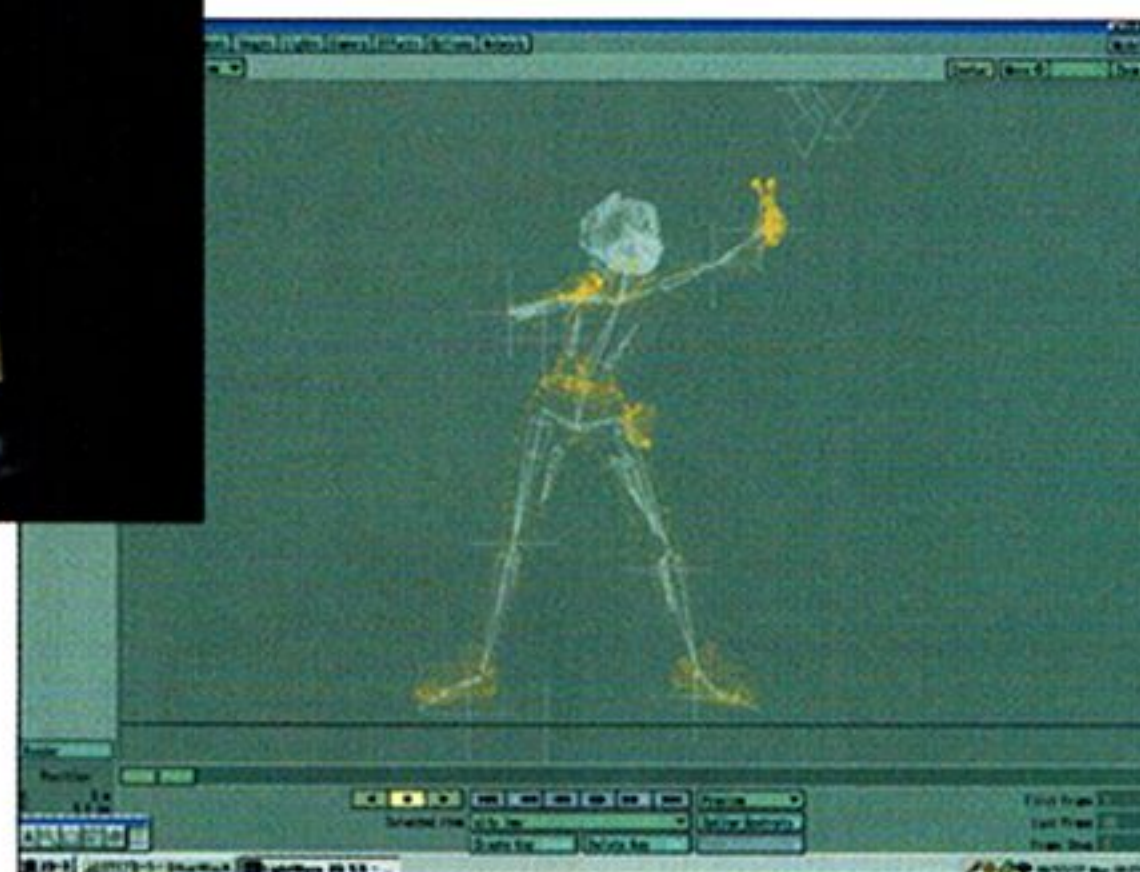
モデリングをするときは，簡単なラフを紙に描いてから作業を始めます。今回はアニメーション用のキャラクターだったので，複雑すぎるデザインは避け，動かしやすいようにデザインしました。

悪ガキっぽいイメージということで，ちょこっとパンクスっぽい感じにして，銀髪・安全靴，腰にはパチンコ用のホルスターを下げているという設定にします。

モデリングは時に困難なこともありますが，一般に楽しい作業のひとつです。実際に顔のモデルの制作過程をご紹介しましょう。

MetaNURBSで顔のモデリングをしているところ

顔のモデルの制作過程

キャラクターをモデリングする際に特に注意していることは，表情やしぐさを伴ったキャラクターを意識するということです。よくCGのキャラクターはツルツルで生命感がないとか不気味だとかいわれますが，その主たる原因はその質感のせいではなく，むしろ表情に乏しいせいだと思います。表情の乏しいキャラクターにはどうしても感情移入しにくいものです。

そこで，表情やしぐさといったものをキャラクターに持たせ，いきいきとしたイメージを演出するために，表情変化などが可能なモデルを制作するのです。

表情のバリエーションを制作しているところ

同様に体も制作します（女性の体をモデリングするときはいつも後ろめたい気分です）。今回は中に骨を入れてポーズを取らせ，それをモデラで多少修正しました。

ボーンIKとモデラーを使ってポーズを作っていく

テクスチャ

先ほど「質感よりも表情だ」といっておいてなんですが，質感にも凝ります。LWにはレンダラにセルシェーダという機能がついていて，アニメのセル画のようなベタッとした色でレンダリングすることができます。今回は少しアニメチックなノリということで，これを使ってもよかったんですが，私はセル画というのはアニメーションの長い歴史のなかで形作られてきた文法だと思うので，無理にCGをセル画風にする必要はないかなと思い，今回はちょっとバタ臭いぐらいのテクスチャにすることにしました（でもセル画風にもそのうち挑戦してみたいです）。

テクスチャは主にPainterで描きました。Painterは昔使っていたMatierと同じような筋のペイントソフトなので，私にとって使いやすいソフトです。

ちょっとバタ臭いテクスチャ

ライティングをしているところ

最終的に決定したアングル

レンダリング

モデルができたら，ライティングしてレンダリングです。モデルのデキがいかによくてもライティング次第では情けない絵になってしまいます。なるべくいろんな方向からライトを当て，見栄えをよくします。また，画面を柔らかいイメージにするため，影はレイトレーシング特有のシャープな影を使わず，シャドウマップと呼ばれるソフトな影を選択しました。

また，少し人形っぽい感じを出すため，被写界深度を再現しています。レンダリング時間はパンフォーカスの場合よりも10倍ほどかかりますが，これも柔らかいイメージを得る助けになりました。

ライティングだけではどうしても希望の色が得られない場合があります。Photoshopなんかで調整するわけですが，その場合ダイナミックレンジを失わないように気をつけます。また，今回は映画のポスターのようなフィルムっぽい質感にしてみました。

Photoshopで調整しているところ

最後に

今回はCGとアニメのあいだぐらいが狙いでしたが，私はその中途半端な性格からか，いつも中間ぐらいが面白いと思っています。たとえば実写とアニメのあいだとか，硬派と軟派のあいだとか。これからも気まぐれになにかのあいだの面白そうなものを作っていけたらと思っています。

筆者ホームページアドレス
http://home.att.ne.jp/red/yosimizu

付録CD-ROM使用法

復刊記念というわけではありませんが，いきなり付録CD-ROMは2枚組です。それぞれのDISCは以下のように分類されています。

DISC1
本誌で解説されているプログラムコード，データなどを中心にされているものです。オーディオトラックが2つついています。

DISC2
必須のDirectX6 SDK並びに開発システムの体験版などを収録しています。

一般的注意

CD-ROMはいわゆるJoliet(Windows95対応ファイル名仕様)で焼かれています。ファイル名が正確に表示されないなどの問題が，使用に支障をきたす場合は，同ディレクトリにアーカイブがあると思いますので，そちらを使用してください。

CD-ROMにはHTMLファイルが内容解説として
Webブラウザをお持ちの方はルートにあるindex.htmをご覧ください。

なお，Windows版のWebブラウザはDISC2DSK6SDK¥EXTRA¥MSIE401¥JPN内に，X68000用のWebブラウザはDISC1の¥MANKAI¥WEBXに収録されています。HTMLファイルが閲覧できない環境の方はREADME.TXTを参照してください。またはHTMLファイルをテキストエディタでお読みになることをおすすめします。HTMLファイルはシフトJISコードで記述されています。

以下では，本文であまり触れられていない収録ファイルについて解説いたします。

DISC1 BLACK DISC

本誌掲載プログラムとデータが収録されております。これは画期的(当誌比)なことなんです。すみません。それら以外に以下のファイルを含みます。

●X FILES

Direct3Dでのプログラミングは今後の定番となると思われますが，プログラムを作るのよりもデータを作るほうがたいへんだという方やDXSDKのサンプルデータだけではつまらないという方も多いでしょう。一般に流通しているXファイルが少なく，3Dプログラム開発の障害にもなっています。そこでCD-ROMにはDirect3D用サンプルモデリングデータが収録されております。

*.Xファイルは，

¥DIRECTX¥D3D¥MESHVIEW.EXE

のように関連付けるのがよいでしょう。

▶どんふり氏の作品群

◀¥DIRECTX¥D3D¥

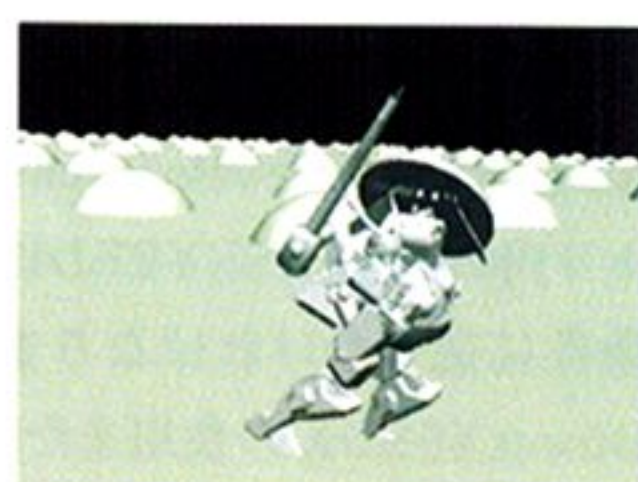
どんふり氏制作のモデリングデータです。制作には主に六角大王が使用されています。全体はひとつのXファイルで構成されていますので，関節データに切り出すなどの加工は各自で鋭意努力してください。

▶腰原氏の作品群

¥MODEL▶

腰原仁志氏の作成されたモデリング＆アニメーションデータです。Lightwave3Dデータとそれからコンバートを行ったものを収録しています。もの凄く重いですのでご注意ください。ディレクトリにはキャラクターごと，パーツごとにデータが収録されています。さらにその下のXディレクトリにDirect3D用ファイルが収録されています。DirectX SDKに付属するViewer.exeで読み込めば再構成することも不可能ではありませんが，単に見るだけの場合はおすすめはしません。Sceneディレクトリにあるデータはアニメーション対応ですので，そちらをおすすめします。なお，Meshview.exeのアニメーション動作は手動コマ送り(Insert，Delete)となっています。こちらは全パーツ独立形式で収録されていますので，根性のある方は関節構造で再構成するなりパーツを入れ換えるなりしてみてください。

こんな重いデータをどうするんだ思われる方もいらっしゃるでしょうが，今後DirectX7でジオメトリ変換ががサポートされれば評価用データとして使用いたしますし(3D Now!評価用だったのだが，結果はいまいちだった)，ポリゴンリデューサなどの開発，Direct3Dモデラ用テストデータなどとしてご使用ください。もちろんLightwave3Dではそのまま読み込んでレンダリング可能です。

Oh!Xでは広く3Dモデリングデータを募集しております。

●Z-MUSIC ver3.02A

X68000用の高性能MML(Music Macro Language)形式総合音楽システムです。X68000シリーズでしか使用できません。内蔵FM音源8音，AD PCM/PCM音，MIDI64チャンネルの演奏などに対応しています。MMLは簡便な記述で高度な表現を可能にするデータ形式です。サンプル曲データをご覧になればおわかりのように(プレーンテキストファイルです)，可読性のある形式で配布されている演奏時間5分弱の曲データが50KBで実現されています。表現力についてはをオーディオトラックでご確認ください。

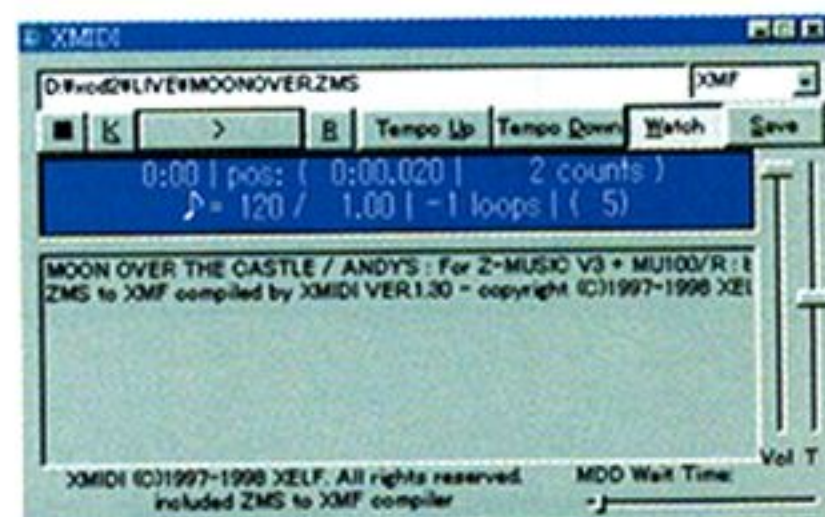
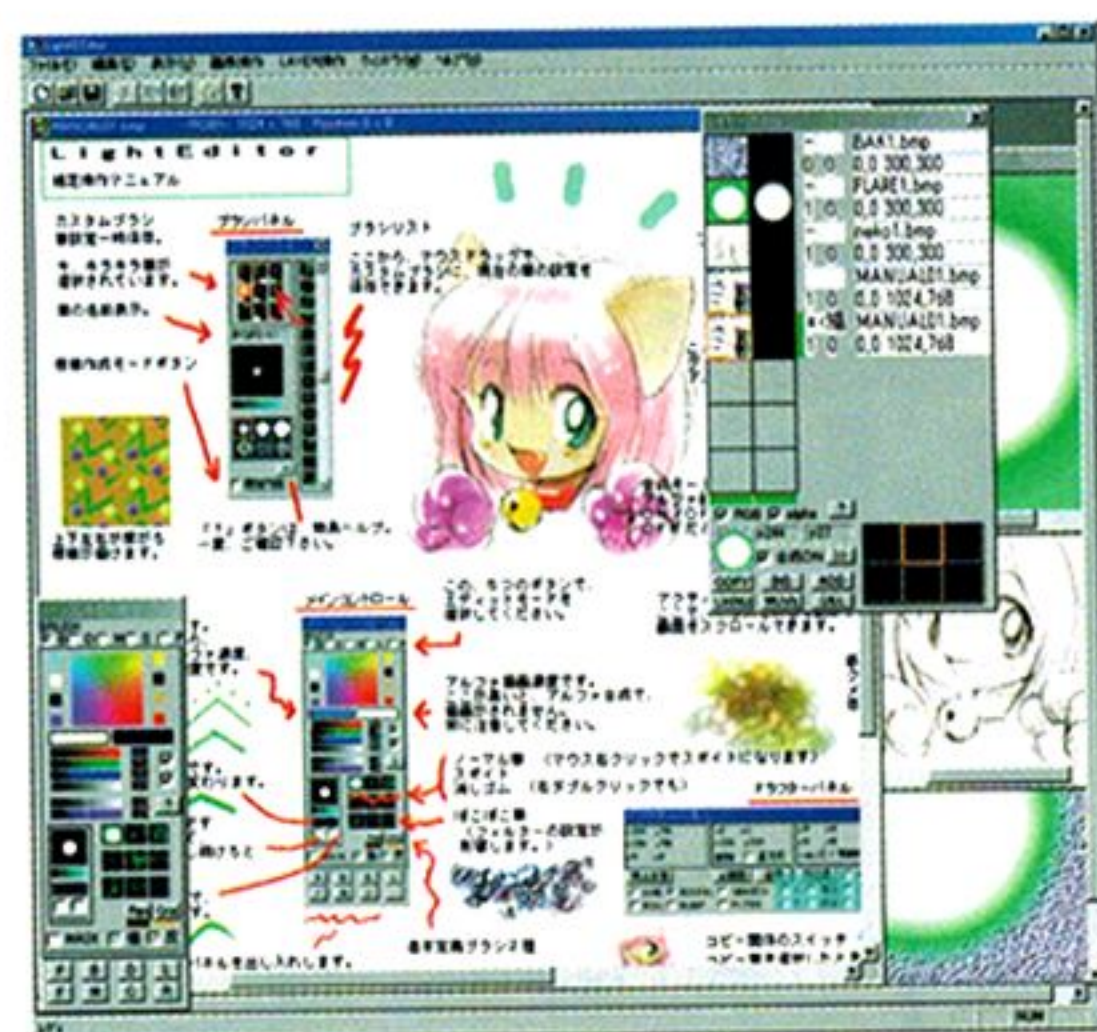
●X-MIDI

XELF氏制作のWindows対応のZ-MUSIC用MIDIデータ演奏ツールです。X68000用ミュージックシステムZ-MUSIC用のソースデータ(ZMS形式)のデータを演奏できます。無限ループなどにも対応しています。これにより，Windows上でZMS形式のMMLを使用してデータ作りが可能になります。

●LightEditor

都築和彦氏謹製のグラフィックツールです。さまざまな独自機能を搭載しています。ワコム製タブレットを使用していない方は，必ず，LED0B.EXEのほうを実行してください。さまざまな機能が複合して動作しますので，慣れないと挙動がつかみにくいかもしれません。¥MANUALディレクトリに詳細なマニュアルがありますので，そちらを参照しつつ試してみてください。なお，LLTファイルを開くときには，必ずファイルメニューの「LLT(LayerList)ロード」をご使用ください。

●EX for Win

以前Oh!Xで制作していたグラフィックツール，EX-SystemのWindows版の最新バージョンです。

XMIDIプレイヤー

Light Editorとそのマニュアル

EX for Winのパターンブラシ

DISC2 WHITE DISC

各種開発ツールなどを収録しています。

●DirectX6 SDK

ドライバをセットアップ後，ドキュメントを読むなり，サンプルを動かすなりしてみてください。初めての方には，¥DOC¥DIRECTX5¥JAPANESE¥HELPがおすすめです。開発環境のない方はドライバだけ入れればよいでしょう。サンプルプログラムはCD-ROMから実行可能です。

●DirectX6 SDK使用上の注意

CD-ROMに含まれるMicrosoft DirectX6 Software Development Kitの権利はMicrosoft Corporationに帰属し，その使用および，これに基づく製品の開発，頒布じゃMicrosoft DirectX Development Kitのセットアップ時に表示される使用許諾書(英語)，およびMicrosoft DirectX6 Software Development Kitを含むディレクトリ内のLICENSE¥REDIST.TXT(英文)の各条件に従わなければなりません。Microsoft DirectX6SDKの使用または使用不能から生じるいかなる損害についてもMicrosoft Corporationは一切責任を追いません。Microsoft DirectX6 Software Development Kitの内容に関するお問い合わせサポートは受け付けておりません。

●3D Aterier体験版

マイクロネットの3Dモデリングツールです。Direct3D Xファイルの編集も可能です。D3D + VBのサンプルを実行する際にDLLが足りないといわれた場合は，この体験版をインストールしてください。確実に必要なファイルがインストールされます。

●Mac版CodeWarrior Lite

Macintoshで代表的な開発環境であるCodeWarriorの体験版です。基本ツール環境とC++言語環境をインストールしてください。

●VisualCafe体験版

Javaアプリケーション作成講座で使用されているWindows用Java開発ツールの体験版です。解説記事を読む際にこおツールでサンプルを参照してください。

●Genesis 3D SDK

Webで公開されているEclipse EntertainmentのWindows用の3Dゲーム開発環境です。使用，配布，製品の販売などに関しては，ほぼライセンスフリーとなっておりますが，これを組み込んだプログラムでは起動時にGenesis3Dロゴを表示してください。

DirectX6ではまだサポートされていないポリゴンの継ぎ目をなくす処理やモーフィングなどがサポートされており，サンプルゲームのソースリストが付属しています。

●TANARUS体験版

Gamer's Eyeで紹介されていたネットワークゲームの体験版です。ネットワークに接続しなくても基本訓練などが可能になっています。もちろん，ネットワークに接続して遊ぶことも可能です。

Hardware Geeks

プリンタいろいろ工房

Text：菊地 功/Isawo Kikuchi

「やべーなあプレゼント出すものないぞ」「じゃあ作りますか」というわけではないんだが，ひそかにプリンタ関係でなんか遊べないかと模索している。今回はプリンタ周辺製品のわりと一般的な使い方を紹介しよう。

カラープリンタは使われているか

CPUやハードディスクなどの躍進はめざましく，それらの進歩のあまりの速さに影が薄くなりがちであるが，実は近年になってプリンタもかなりの発展を遂げている。それもフルカラー。

いまでこそカラーインクジェットプリンタなど，モデムの次に買い揃える周辺機器として一般に定着しているが，一昔前は多階調のカラープリンタなど，一部のブルジョワだけが所有できる高級な周辺機器だったのだ（もっとも，パソコン自体もブルジョワの娯楽道具であったが）。いまから思えばその頃のカラープリンタなど，高い，汚い，遅いと三拍子揃ったシロモノであったが，それでもみんなカラーを選んでいたものだ（一時期からは熱転写だったからまだ安かったんだけど）。

dpiうんぬんが一般ユーザーで語られる以前の，A4サイズ1枚印刷するのに30分も1時間もかかるような，そんなものを20ウン万円で買ってウハウハいっていた頃もあった。それがいまや下手をすると1万円台で，そのころの何倍も綺麗で何倍も速いものが買えてしまう。

現在では価格競争のほうが激しいが，昨年までは品質競争が白熱していた。特に品質を大きく引き上げる切っ掛けとなったのは，一昨年のフォトインクの登場だろう。その年の暮れにはエプソンがクオリティで他社を大きくリードするPM-700Cを投入し，エプソンの牙城を築く礎となった。筆者も昨年になって辛抱たまらなくなり，購入に踏み切ったのだが，筆者はA3の魅力に取りつかれてPM-2000Cの購入となった。

カラープリンタ否定派からは，まず第一に「なんに使うの？」という質問を浴びせられる。「目的などない。カラープリンタは男のロマンだ！」などといってみたところで，実際に使われることがなければただのがらくただ。プリンタなどは，喜び勇んで買って3日が華というのは筆者も重々承知している。とりあえず手持ちのCGを印刷してみて，それに飽きたらインターネットからエッチな画像をダウンロードして印刷してみて……なんてやっても，すぐに飽きるのは目にみえている。

まあ，確かに1万数千円で買ったようなプリンタならば，消耗品感覚で「年に一回年賀状シーズンに活躍すればそれでいーか」ってなことになるのかもしれないのだが……。

そこでだ，実際にカラープリンタを紙以外に印刷して活用する方法を探してみよう。幸いなことに，最近ではカラープリンタを活用するためのいろいろなキットがいろいろなメーカーから出回っている。そういったものを模索して，使い道を考えてみようじゃないか。「買えば使い道はあとからついてくる」が私の（業界の，か？）モットーなのだ。

アイロンプリント

ありがちである。インクジェットでも比較的水に強いキヤノンのキットが割と昔からあるが，いまではEPSONもアイロンプリントペーパーを出している。なお，ALPSのMDシリーズのほうが，ドライインクを使っている分，洗濯時の信頼性は高いが，クオリティではPMに優るものはない。

まずアイロンプリント用紙に絵柄を出力する。このとき，左右反転させておかないと，いざ転写したときに，柄が裏返しになってしまう（って普通その前に気づくよな）。出力したら，余白部分を切り取る。別に切り取らなくてもいいんだけど，アイロンプリント紙は全面に糊が載っていて，当然ながらインクの載っていない部分の糊も転写されるので，切り取っておかないと余白部分も糊でパリッとなる。そうしたら，転写先の布の上に用紙の印刷面を下にしてレイアウトし，上からアイロンを当てる。ここはかなりしっかりと，特に縁は剥がれやすいので念をいれよう。

あとは用紙を剥がすだけなのだが，これがちょっとムズい。無造作に用紙のエッジに爪を引っかけて剥がすと，糊まで剥がしてしまう。ここは用紙をスライスするようにして用紙の側面に爪を入れるとよい。あと，用紙を剥がすとき，力任せに引っ張らないこと。もし熱の加え方が足らず，糊が破れて用紙についてきてしまったら，もう一度用紙を当てて，上からアイロンを当て直そう。

うまく剥がれれば完成だ。Tシャツ辺りがオーソドックスだが，ハンカチのワンポイント（吸水性はなくなりそうなので，全面は止めたほうがいい）なんかもいいだろう。プレゼントとかイベントなんかに結構お手軽なんではないだろうか。

マウスパッド

キヤノンなどは自前でキットを出しているが，ここでは株式会社きもとの「マウスパッド工房」を紹介しよう。こいつを選んだのは，金・銀フィルムのキットがあったからだ（もちろん白もある）。要はこの金・銀のフィルムの上にプリンタで印刷

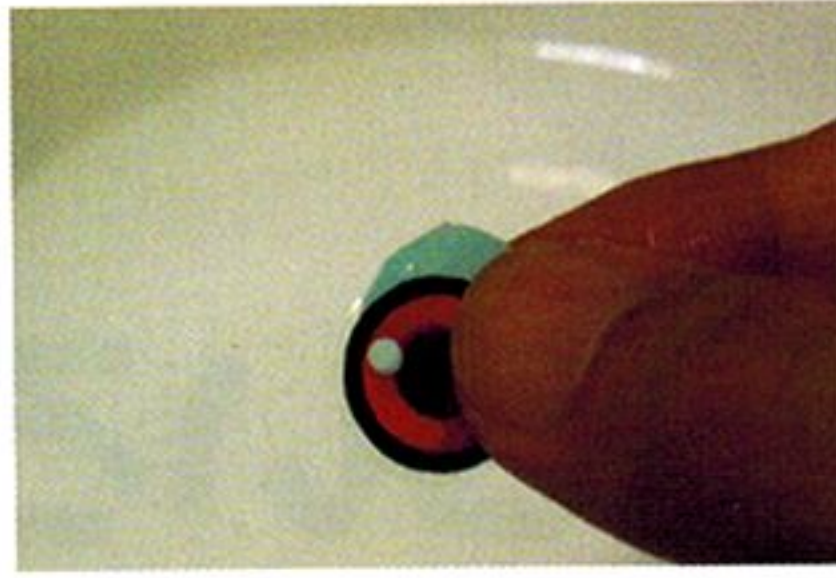
プリンタ活用のための製品がいろいろ出ている

打ち出すときには反転させておくこと

オーソドックスにアイロンプリントをしてみる

はがすときは慎重にしないと皺がついたりする

A4サイズだとだいたいこんな大きさになる

今度はマウスパッドを作ってみた。意外にいい感じ

クリアデカールでピンポン玉に目を入れる

曲面には綺麗に貼れないので，デカール軟化剤を使う

したものを，マウスパッドにできるのだ。

まずはフィルムに普段と同じように印刷を行う。次いで，上からカバーフィルムを貼る。これは，印刷面を保護すると同時に，マウスの滑りをよくするためのものだ。その後，パッドの基本サイズ220mm×180mmにカットし，パッドに張ってからコーナーをカットして完成だ。これがなかなかどうして美しい。金・銀なんて成金趣味にならんかいなと思ったりもしたが，カバーフィルムの表面のざらつきが適度に乱反射し，下品にならない。

ところで，このキットには金と銀のフィルムが2枚ずつ入っているが，パッドは1枚しか入っていない。印刷ミスの予備というわけではなく，実はこのフィルムは一度貼っても何度でも剥がして貼り直せるのだ。つまりこのマウスパッドは4枚の替えフィルムがあり，その日の気分によって絵柄を替えられる……って，いちいちそんなことするやついるのか？　ま，剥がせるわけだから，直接デスクに貼ったりしてもいいし。その他，株式会社きもとは「ホワイトボード工房」なんてものも出している。その名のとおりホワイトボードで，作り方は「マウスパッド工房」と基本的には同じだが，こちらには冷蔵庫なんかに貼れるマグネットタイプと，弱粘着タイプのフィルムが1枚ずつ入っている。イレーザー付きの専用マーカーまで入っている芸の細かさだが，それならマーカー用のトレイもほしいところだ。

デカール

よくプラモデルとかについてる，俗に水シールとかも呼ばれてるやつだ。紙の上に薄い糊付きの透明皮膜が乗っており，水に浸けて皮膜を遊離させたあと，対象に乗せるシールである。このデカールが通常の透明なセルロイドのシールと比べて優れている点は，厚みが薄いためにシールの縁が目立たないこと，凹凸のある面にもフィットしやすいことだ。WAVEからプリンタで印刷できるデカールが発売されているが，一度水に浸す関係上，インクジェットプリンタではなくMDシリーズ専用である。試しにPM-2000Cで印刷してみたところ，水に浸す以前に用紙にインクが乗らずにダマになってしまった。幸い筆者はMD-4000Jも所有しているので，それで印刷してみよう。

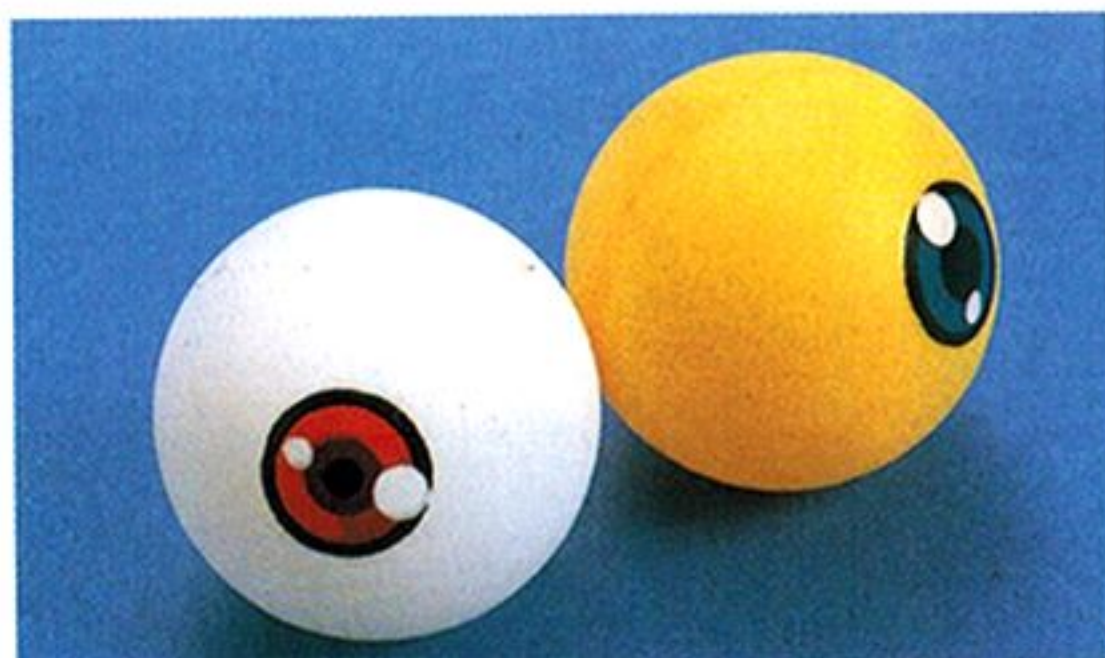
印刷に際しては，特に特殊な手法は必要ない。普通に紙に印刷するようにプリントアウトすればよい。その後，模様の縁に沿って余白を切り，数秒間水に浸す。あまり長い間浸していると糊が溶け出てしまうので，注意が必要だ。十分に浸ったら，軽く擦るようにしてデカールを台紙から剥がし，そのまま対象に乗せる。いったん乗せてからも，スライドさせて移動させることができるので，正確な位置あわせは乗せてからピンセットなどですればよい。ただ，破れやすいので，あまり無理に引っ張ったりしないこと。

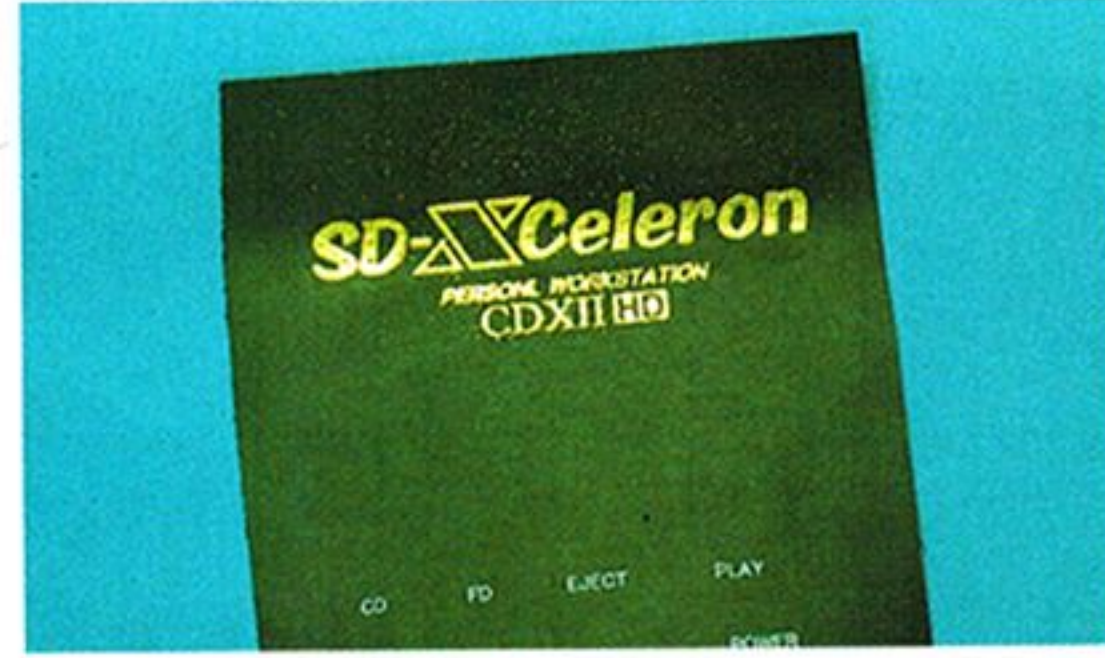
あと，もし凹凸のある面にしっかりフィットしない場合は，模型店に売っているデカール軟化剤を使うといいだろう。最後にティッシュなどで軽く押しつけて完了だ。完全に乾くまで触らないように注意しよう。また，乾いたあとでも強く擦るとはがれてしまうので，できるだけ触らないようにするのが望ましい。

ところでこのデカール，透明であるので，地が白以外の場合には思ったように発色しない。黄色い地に青いデカールを貼れば緑になってしまうのは想像に難くないだろう。このため，WAVEでは特色白を下地として印刷することを推奨している。特色白はMD-1000以降の機種から対応したインクカセットなので，普通ならそれ以前の機種であるMD-4000Jでは利用できない。しかし，簡単な細工をすることで，MD-1000以前でも特色白を使うことができるのだ。そのあたりのことは私のホームページ特別室，http://www.win.ne.jp/~kisawo/gareki/に説明があるので，興味のある方は見てもらいたい。

なお，WAVEからはあらかじめ全面に白で下地を施したデカールも販売されている。こちらを使えば特色白での印刷は不要になるが，模様の縁を綺麗に切り取らなくてはならないので，デカールのメリットは半減である。

最後に

このほか，少々季節は外れるがうちわのキットとか，話によると印刷できるスクリーントーンなどもあるらしい。グラデーションなどはもちろん，デジカメで撮った写真に網点フィルタでも施して印刷すれば，風景や建物のスクリーントーンなど自由自在だ。また，現行製品ではないが，緑電子のPRN-21という昇華型のフォトプリンタは，オリジナルのマグカップを作ることができる。いまでもときどき在庫処分で1万円を割る値段で売っていたりするので，興味があれば探してみるとよい。マグカップキットは店頭で見かけることはまずないが，緑ダイレクト(http://www.midorielec.co.jp/)から直接購入することができる。そのほかにも，探せばきっとプリンタのさまざまな使い道というのはあるはずだ。せっかく買ったのだ。パソコンラックのてっぺんで埃を被らせておくだけでなく，積極的に使ってあげよう。

できあがるとこんな感じ

やはりデカールのメイン用途はこの辺りか？　川原氏の描く女の子の目を入れた人形

番外編。プリントゴッコと組み合わせてメタリックな立体バッジを作る

TANARUSに学ぶゲームバランス学

© 1997,1998 Sony Interactive Studios America Portions ©1996,1997,1998 Pyrotechnix

西川善司

チームを組んでの戦車による団体戦。TANARUSは完全日本語対応した数少ないネットワークゲームだ。日本ではまだ普及していないネットワークゲーム普及の口火になるかもしれない作品として注目される。ゲームバランス以外にも今後参考にすべき点は多いだろう。

ネットワーク対戦ゲームの面白さとは?

今年の夏，ちょっと面白いネットワーク対戦対応の3Dシューティングゲームの日本語版が発売された。その名は「TANARUS(タナラス)」。

TANARUSはもともとはアメリカのSony Interactive Studios Americaで開発されたゲームで，本国では継続的に好評を獲得しているタイトルだ。もういまさら「ネットワーク対戦ゲーム」というキーワード自体にはそれほどの目新しさはない。TANARUSは，それよりもそのゲームバランスの設定が絶妙である点が高く評価されているのである。

実際，人気を博したアクション系のネットワーク対戦ゲームを思い返すとすれば「DOOM」「Quake」シリーズ，などが筆頭に挙げられることになるわけだが，これらを「ゲームバランス」という観点から評価するとあまり手放しで誉められたものではない。強力な武器を手に入れたプレイヤーと出会えばその瞬間に勝負がつくことも多く，状況によっては「なにがなんだかよくわからないうちにやられている」ということがよくある。

それでもこれらのゲームが楽しかったのは，「戦っている相手が自分と同じ人間(が動かしているキャラクター)」だという点と，「いつ出合い頭に殺されるかわからない」という「緊張感」の部分にあったといっていい。

TANARUSでは，こうした，あっという間に勝負がつくギャンブル的なゲーム性を極力排除し，リアルタイムの3Dアクション(シューティング)ゲームという形態を取りながらも「ストラテジー(戦略性)」要素を多分に盛り込んだゲームデザインがなされている。

そしてその「ストラテジー」を具現化するために「チームワーク」をプレイヤーに作り出させ，これがさらにプレイヤー間に「コミュニケーション」という副次的なフィーチャーを生み出させているのである。

戦略性とチームワーク，そしてコミュニケーション

TANARUSの面白さは「戦略性」「チームワーク」「コミュニケーション」……この3つのゲーム性要素に集約される。

「戦略性」とは敵の行動を予測して，これにさらに打ち勝てるような行動方針を練り上げることだ。こうした要素は当然のことながら「AGE OF EMPIRES」(マイクロソフト)に代表されるようなネットワーク対戦ストラテジーゲームには必然的に存在するわけだが，TANARUSのような3Dシューティングゲームに盛り込まれている例は少ない。

そして「チームプレイ」に対応した対戦ゲームは現在では少なくない。知ってのとおり，Quakeシリーズにもチームフラッグ争奪戦モードはある。ところが，不思議と「チームプレイ」に対応したほとんどのゲームで「チームワーク」が生まれることがないのである。いったいなぜなのだろうか。

まず，Quakeシリーズでは，各プレイヤーが獲得した武器によっては最強状態となり，単独行動を行っても勝ててしまうことに原因があると思われる。「チームワーク」が生まれるためには「チームワークを生かしたプレイを行わなければゲームに勝てない」……というゲームデザインでなければならないのだ。TANARUSはそのあたりをよく考慮して各自の戦力のバランスを取っている。

そして，チームワークを実践するためには，各プレイヤー間の意思疎通が行えなければダメだ。実際の軍事行動と同じく，個々の戦力がシステマティックに行動するにはお互いの状況の情報伝達が必要不可欠なため，自然とTANARUSではゲームプレイ中のチーム内チャットが多くなるのである。TANARUSでは戦場に参加している全員へのメッセージ送信のほか，自分のチームメンバー向け専用のメッセージチャンネルがあり，日本語版ではFEPを使って漢字かな交じり文のメッセージも送受可能となっている。TANARUSの「コミュニケーション」というフィーチャーはゲーム設計者がプレイヤーに押し付ける要素ではなく，「戦略性」と「チームワーク」を実行するための必要手段として生まれているものなのである。

やや前置きが長くなったが，TANARUSがいったいどんなゲームかを紹介していくことにしよう。

TANARUSの基本ルール

プレイヤーはレーザー兵器を装備可能な近未来の戦車に乗り込むことになる。この戦車は電気動力になっており，レーザー砲を撃ったり，移動したりすると電池を消耗する。つまり活動範囲に限界があるということだ。

戦場には赤色，青色，緑色，灰色の4色の軍勢があり，プレイヤーはそのうちのひとつに属してプレイすることになる。現在のバージョンではひとつの軍勢(チーム)は5人まで参加することができ，すなわちひとつの戦場で，5人×4チーム，同時に20人が参戦できることになる(10対10の2チームモードもあり)。

それぞれのチームの本拠地には将棋でいうところの「王将」に相当する「チームフラッグ(軍旗)」があり，これを敵チームの本拠地に持ち去られるとその時点でゲームオーバーとなる。つまり自軍のフラッグを守りつつ，敵のフラッグを奪うということがゲームの本質的な目的ということだ。

さて，戦場には初期状態で「前線基地」と呼ばれる，各戦車が補給可能なポイントが点在している。レーダー上に白色の□マークで表されており，ここに到達するとバッテリーの充電，弾薬の補給，そして後ほど紹介する，オプション装備の補充/交換までが行えるのだ。

各戦車にはバッテリー容量という活動制限が課せられているといったが，この前線基地を利用していけば，戦場のあらゆるところに行けることになる。

ところが，この誰でも立ち寄れるコンビニエンスストア的な前線基地を，自軍だけが利用できるように改造してしまうことができるのだ。これが「前線基地の占拠」という行動だ。占拠は「前線基地占拠衛星」という人工衛星を，自軍の本拠地から発進させて行う。占拠には30秒ほどの時間がかかり，占拠行動中にその衛星を敵に撃ち落とされてしまうと占拠は失敗となる。

占拠に成功すると前線基地は，レーダー上にそのチームカラーで彩られた□マークで表されるようになり，その軍に所属する戦車にしか利用できなくなるばかりか，近づく敵軍戦車の電力を(微少ながら)奪う能力を持ってしまう。

つまり，戦場に点在する前線基地を効率よく自軍のものにしていくことが，自軍の行動範囲を広

最前線は乱戦模様。敵味方入り乱れての自然の撃ちあいが始まった

日が沈みかけてきたところ。TANARUSではちゃんと太陽が動いており，日が昇りそして沈む

完全に日が沈み，あたりは暗闇に包まれる。視界が非常に悪くなるが，それは相手も同じこと。奇襲の絶好のチャンスなのだ

味方戦車が敵戦車に背後から撃たれている。援護しなければ。味方戦車はメインスクリーン上で水色の枠で囲まれる

げ，なおかつ敵の行動力を衰弱させる効果があるということだ。

占拠されてしまった前線基地も，自分たちの占拠衛星を誘導させれば占拠の「仕返し」ができる。

しかし，作戦行動の重要拠点となる前線基地をそうやすやすと譲り渡してくれるはずもない。そう，ここで，ゲームの本質的な目的である「敵軍旗の略奪」を実行する前段階として，前線基地の攻防が発生するわけだ。

新たな前線基地の確保に行くか？ それともすでに占拠した前線基地を守るか？ このあたりもTANARUSの重要なゲーム性になっている。もちろん，2つの行動は自分ひとりでは同時にできないので，ここでチームメイトとの協力，役割分担が必要になってくるのだ。

なお，基地占拠用の人工衛星の誘導装置は自軍の本拠地のみで入手可能となっているので，とある前線基地を拠点にして次々に周囲の前線基地を占拠していくことは行えないようになっている。また，敵の軍旗を持ち去るための軍旗回収装置も自軍の本拠地のみで入手可能だ。敵軍本拠地にもっとも近い前線基地を確保したとしても，軍旗を回収する装置は一度自軍本拠地まで取りに戻る必要があるわけだ。

コミュニケーションの自然発生

この「取りに戻らなければならない」という要素はいかにもゲーム的だが，ここにもTANARUSの巧妙な「仕掛け」が潜んでいる。

たとえばある基地の占拠をめぐって攻防を繰り広げているとき，敵が一瞬ひるんで目的の基地の防備が薄くなったとする。このとき基地を占拠したいわけだが，全員が占拠装置を本拠地に取りに戻ってしまってはせっかく押しのけた敵をまた呼び戻してしまうことになる。

ここでチーム内の誰かが「誰か前線占拠装置を持ってきてくれ」と発言する必要性が出てくるのだ。これはもちろんチーム内専用のチャンネルを使ってのメッセージ送信になるわけだが，チーム内のメンバー間にこうした情報伝達の必要性を間接的になおかつ自然に行わせるゲームデザインはうまいとしかいいようがない。

一度こうした連携行動が成功すると，次の作戦行動に向けてそのチーム内にはおのずと「チームワーク」が生まれてくるようになる。不特定多数の人間がランダムに参加するネットワーク対戦ゲームにおいて「チームワーク」が自然発生するというのはこれまでのリアルタイムタイプの対戦オンラインゲームからは考えられなかったことだ。

そして，「最寄りの自軍前線基地」ではなく「『わざわざ』本拠地まで取りに戻る必要がある」というのも意味のあるゲームバランスの設定といえる。一度本拠地に行って戻ってくるという行動はそれなりに時間を要する。本拠地に戻るということはつまり，前線での戦いから一定時間外れるということになる。すなわちその軍勢はその間，数的にひとり分戦力を失うことになるわけだ。つまりこの間，敵側にすれば有利になるわけで，後退させられた戦線を一気に押し戻すきっかけになるかもしれないのだ。

戦況を自軍に有利な方向へと導こうとするとそれなりにリスクを背負うことになる……こうした「駆け引き」の設定は戦略ゲームでは欠かせない要素だ。

単純な仕掛けと要素の組み合わせだが，これがチームワークを発生させ，そのチームワークが戦略を生み，さらなる戦略がチームワークを育む……TANARUSではこういうゲームシステムが潜在的に仕掛けられているのである。

5タイプの戦車に見るゲームバランス

TANARUSでプレイヤーが搭乗できる戦車は全部で5種類ある。それぞれが見た目の形の違いだけでなく，装甲の厚さ，走行スピード，標準装備の兵器が異なり，特徴づけがなされている（52ページの写真参照）。

そして各戦車には「拡張スロット」が設けられており，用意されている約40種類のオプション装備をプレイヤーの好みにあわせて自由に搭載することができる。オプション装備は戦車各機種の弱点を補うもの，あるいは特長を増強するもの，はたまた特殊機能を追加する役割を持っており，ここにもTANARUSの面白さが潜んでいる。

各オプション装備はそれぞれ占有するスロット数が異なり，強力な，あるいは特殊な効果をもたらす装備ほど占有するスロット数が多くなっている。そして戦車各機種は機種ごとに用意されているスロット数が異なるため，あれもこれも詰め込むようなことはできなくなっているのだ。

たとえば，前述した，前線基地を占拠するための人工衛星を誘導する装置は占有スロットが6だ。スロット数が，少ない機種の戦車で5，多いものでも10であることを考えると，これを装備できない機種もあるし，装備できる機種についても過半数のスロット数をこのユニットに占有されてしまうことになり，戦いに不可欠な防御システムや電力増強システムなどを装備できないことになる。これを装備したときのその戦車単体の戦力は著しく劣ることになるのだ。

各プレイヤーは自分の戦車になにを装備するかを思い悩むことになるのだが，結局，「最強の組み合わせ」というものがTANARUSの戦車には存在しないのだ。特殊機能を付加すれば基本性能が落ち，基本性能を増強すれば特殊機能は実装できない……このようなジレンマが発生することになり，搭乗戦車の設定はどのように行っても味方のバックアップあってこそ生きてくるように

灰色軍の戦車が緑軍の戦車を追いたてる。後退する緑戦車はうかつにも背後に仕掛けられた地雷を踏んでしまう

薄い半透明のフィールドが機体を覆っているのがわかるだろうか。これがシールドだ。攻撃によりこれを剥がされ，さらに機体の装甲を撃ち抜かれるとその戦車は破壊される

衛星が前線基地を占拠しているところ。ビームの照射が完了するとその基地の占拠が完了する

なっているのである。

結局，誰がどういう装備で出撃するか，というのはチーム内で協議した戦略に準じて行わざるをえなくなるのである。

電力と衛星の妙

TANARUSに登場する戦車は「電力で動く」ことはすでに述べた。この「電力」にまつわるゲームデザインにもTANARUSは緻密なバランス設定が施されている。

各戦車は，もし電池を使いきってしまっても前線基地に戻れば再充電が可能だ。また，回復度は低いが基地周辺でも微量ながら充電ができるようになっている。逆に，前線基地は近づく敵の電力を吸収する機能も持ち合わせている。

これは基地周辺にいれば電力の消耗を最小限に抑えられるということであり，前線基地へ攻め込む側よりも守る側のほうが有利だということである。

このままでは戦況は動かないようにも思える。ところが，ここに攻め込む側としてはリスクを背負いつつ戦いを有利に運べるかもしれない決め手があるのだ。

それが「人工衛星」だ。

ここまでに，前線基地を占拠するときに使う人工衛星の話をしたが，TANARUSにはもう1種類の人工衛星がある。それはオプション兵器としての人工衛星だ。

普段，この人工衛星は自軍本拠地の上空に浮いていて，本拠地に近づく敵を撃退する機能を果たしている。手っ取り早くいえば，普段この人工衛星は，本拠地にある軍旗を警備している……ということである。

この人工衛星は非常に優秀で，ステルス機構を装備した戦車であっても射程範囲であればその位置を算出し，威力はそれほどでもないがレーザービームを発射してくれる。

実はこの人工衛星を最前線に移動させることのできる「衛星移動装置」というオプション装備があるのである。この装置を装備した戦車は，任意の位置へこの人工衛星を最大4個まで移動させることができるのだ。

最前線に移動した人工衛星は，上空から射程距離内にいるステルス戦車を含む敵全戦車に対して砲撃を開始する。人工衛星は，対等条件では攻め込む側に不利な前線基地の攻略を手助けしてくれる非常に優秀な味方となるのだ。

この移動可能な人工衛星の能力は援護射撃だけではない。なんと衛星直下付近にいる味方戦車に対して，微量ながら電力を供給する機能まで持ち合わせているのだ。

攻め込む側は，奪いたい敵前線基地に近づけば近づくほどその基地から電力を奪われてしまうことはすでに述べたが，もし，この人工衛星をその基地近辺へ移動したとするとどうなるのだろうか。そう，この電力吸収効力を弱める（移動させる衛星の数によっては無効化する）ことができるのだ。

もちろん，守る側も衛星を移動することはできるので，結局両軍ともに衛星を戦闘の最前線に持ち込んだときには再びイーブンな条件となり，守る側のほうが有利ということになる。

ところが，思い出してほしい，人工衛星は，軍旗を守っていた……という事実を。そう，最前線に人工衛星を移動させてしまうとその分，本拠地の警備は手薄になるのである。

つまり，とある基地の攻防を囮作戦にして，敵に人工衛星をそこへおびき出させ，警備の薄くなった本拠地へ奇襲をかける……といった戦略もできるのだ。

衛星を前線へ移動させて味方の戦いを有利に運ぼうとすれば，本拠地の警備は薄くなる……というジレンマが生まれるわけで，ここにも「駆け引き」があることになる。

デスマッチ対戦に飽きたネットゲーマーに捧ぐ

いくつかの明快なゲーム要素の下にぶつかりあう勢力のそれぞれの立場に対し，有利不利を設けながらもその合算値は0（イーブン；対等）近くにまとめているところにTANARUSのゲームバランスのうまさがある*1。

そして，その「イーブン」の均衡を破るために「戦略」が生まれ，この戦略を実行するために「チームワーク」が発生し，チームワークを実現するためにプレイヤー間の「コミュニケーション」が生まれる……こういう図式がTANARUSには成り立っているのだ。

*　*　*

これまで，コンピュータゲームのゲームバランスというものはCPU対人間プレイヤー，あるいは人間の1対1を想定して考えられてきた。しかし，ここにきてネットワーク対戦ゲームという新たなゲームシーンが生まれ，複数人による人間対人間のプレイが本当に楽しいゲームデザインとゲームバランスが望まれている。TANARUSはそんななかで生まれた，いまだ数少ないよくできた具体例といえると思う。

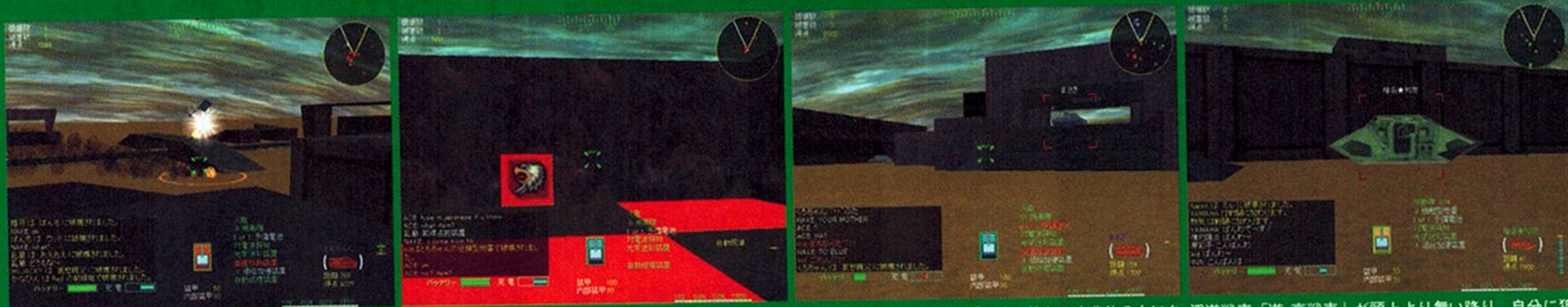
ただ相手を殺して生き残ればいいというだけのデスマッチ型ネットワーク対戦ゲームとはひと味違った，「いつも頼れる味方が近くにいる」という一種独特の連帯感を味わえるプレイ感覚と，「行動1つひとつを常に『戦略』に結びつけていかなければ勝てない」という知的な緊張感を味わえるのはまだこの「TANARUS」だけだ。まだ，プレイしていない人は一度試してほしい。

*1　ただし，リアルタイムゲームであるために，プレイヤーアビリティの格差により，設けられた「均衡」が予想外に傾くことがある。結局，単純な数値パラメータだけのゲームバランスではなく，幾度となく繰り返されたテストプレイによって生まれた「均衡」であることは間違いない。ゲームバランスはゲームデザインの「質」にも左右されるが結局最終的には「テストプレイ」によって調整されるものなのだ。

赤軍戦車が立体的な陣形を組んで敵基地に向けて一斉射撃。こうしたチームワークが自然と発生するのがTANARUSの魅力

補給中の戦車を火砲で遠距離砲撃，見事命中。敵が周辺にいるときは補給中も気を抜けない

衛星を引きつれていざ出陣。最前線での戦いを有利に導くことができる強力な味方となるがその分本拠地の防備は甘くなる

画面中央に見えるのが敵前線基地。画面左下に見える味方戦車とともに進軍しているところ

誘導ミサイルがステルス戦車を直撃。砲身が飛びあがる大爆発。戦車はダメージを受けると爆煙をあげるようになる。それはステルス戦車とて同じ。移動する爆煙があればそこには手傷を負ったステルス戦車がいることになる

これが軍旗(写真は赤軍のもの)。これを敵に奪われ本拠地まで持ち帰られるとゲームオーバーとなる

前線基地は写真のような「やぐら状」の建造物の中にあることが多い

浮遊戦車「遊-高戦車」が頭上より舞い降り，自分に狙いを定めたところ。地雷しか装備していないプレイヤーの戦車は絶体絶命

戦車解説

雷-軽戦車 (Lightning)

高速移動を得意とする戦車。一撃離脱の戦法が得意だが，砲台が左右180度回転できる特性を利用し，1対1の対決になったときでも，相手の周りを回りながら攻撃したり，前方に走り去りながら後方へ攻撃するなどのトリッキーな戦法で戦うことができる。その高速性を活かして軍旗回収や占拠活動を行うこともできるだろう。

弾道弾発射装置を標準装備しており，誘導ミサイルなどを発射することもできる。車高が低いので中戦車や重戦車の弾を食らいにくいという隠れたメリットもある。

鋭-中戦車 (Vanguard)

TANARUSにおける主力兵力的存在。スピード，装甲の厚さ，オプション兵器搭載用スロットの数……すべての要素で中間的な性能を持つ。特に弱点がない代わりに突出したところもないので初心者向きの機体だ。逆に装備するオプション装備によってさまざまな特徴づけも可能なので上級者にとっては「いじりがい」のある機体となる。

光線砲ではもっとも強力な「光線砲四型」を標準装備しているのでスロットに装備するオプション装備は防御系のものを中心に装備することになる。

剛-重戦車 (Devastator)

装甲が厚く自動修理装置を装備しているため時間とともにダメージを回復する能力を持つ。強力な防御性能の半面，動作速度は遅い。また小回りも利かず，そのボディも巨体なので狭い場所での戦闘は逃げ場を失いがちになる。搭載スロット数は全機種中最大なので，スロット数を大量に消費する前線基地占拠用衛星の誘導装置はこの戦車に装備させることが多い。

また，焼夷弾などを発射できる火砲を標準装備しているため遠距離砲撃作戦には欠かせない機体となる。略称「デブ」。

忍-隠密戦車 (Chameleon)

標準ステルス機構が組み込まれており，レーダーおよび視覚から完全に姿をくらませて行動ができる。移動スピードも速いが，装甲は薄い。その名のとおり，主に偵察任務や地雷設置任務に適している。攻撃行動時は一瞬だけステルス機構が無効になるためこのときに見つかって砲撃されるとひとたまりもない。また，被弾中もステルス機構が解除されてしまう。常に敵の裏の裏をかくことを念頭に操縦する必要がある。

波動砲を標準装備しているがエネルギーの充填に時間がかかるため，接近戦には向かない。非常用兵器として認識すべき。略称「カメ」。

遊-高戦車 (MagRider)

電磁浮遊装置を装備しており，高いところから落下しても機体にダメージを受けることはない。浮遊しているため照準が狙いにくく，また砲台が回転しない欠点もあるが，逆に平行移動をしながらの攻撃が簡単にできるという独特な動特性を持っており，操縦の仕方によっては非常に強力な機体になりうる。

射程距離の長い光線砲二型を標準装備しているので遠距離からの衛星破壊が得意だ。垂直上昇ユニットを搭載して空から地雷を落とすといった変則的な攻め方もできる。

火砲などの衝撃の強い火器を食らうと浮遊している関係で機体がよろめきやすく，あまり接近しての撃ちあいには向かない。

column 汚いプレイ?

対戦ゲームで必ず議論に上る「汚いプレイ」という話題。TANARUSをプレイしていると相手のプレイに対して「卑怯！」とか「汚い！」という罵声が出ることがある。

しかし，TANARUSはゲームバランスの設定が絶妙であるため，いまのところこうした罵声はほぼすべてが「弱者の遠吠え」にすぎない。よく起こる状況について本当に「汚い」かどうか検証をしてみよう。

火砲がずるい

レーザー兵器メインのTANARUSではあるが，焼夷弾，徹甲弾，破裂弾といった弾薬を使用する砲撃手段もある。これがTANARUSでは「火砲」という分類がなされており，これを「汚い」といわれるのである。

どこが「汚い」かというと，まず，レーザー兵器が直線的な弾道を描くのに対して火砲は放物線を描いて飛んでいくため物陰に隠れた敵にも命中するという点が挙げられる。そして焼夷弾，破裂弾は着弾すると周囲に爆風が広がりこれに触れてもダメージを食らう，シールドを突き抜けて直接機体の装甲に被害を与える，という点も「ずるい」といわれる。

はたしてそうだろうか。

まず，弾道が放物線を描くというのはメリットでもありデメリットでもある。照準器が役に立たないため弾を敵に命中させるのが難しいのだ。周囲にもダメージを与えられるとはいってもそのダメージは微量だ。さらに，その爆風は周囲の味方戦車にまでダメージを与えてしまう点も欠点として挙げることができるだろう。

またレーザー砲に比べて弾速が遅い点，連射速度が遅い点も火砲の欠点だ。したがって火砲は遠距離射撃には向いているものの，その砲撃が予測されて懐に飛びこまれての接近戦では不利きわまりない。

そしてバッテリーが許す限り弾数が無制限なレーザー兵器に対して火砲はその数は有限だ。貴重なスロットを消費して砲弾をストックしておく必要があるのだ。砲弾を使いきればその戦車はまったく攻撃手段を失うことになる。また火砲は(火砲標準装備の重戦車を除き)専用の発射装置をこれまたスロットを消費して装備しなければ発射すらできない。

メリットよりもデメリットのほうが明らかに多い火砲は，使うほうにも勇気がいることになり，これでやられたプレイヤーの罵声はただのいいがかりにすぎないということになる。

ステルス戦車がずるい

忍-隠密戦車はステルス機能を持っておりメインスクリーンはもちろん，レーダーからも察知されずに行動が可能となる。それ以外の戦車でも光学迷彩装置と対電波装置をオプションスロットに搭載すれば同等のステルス機能を持つことができる。

こうしたステルス戦車にやられると「ステルスは卑怯」という発言が出ることがある

しかし，このステルス機能は，シールド機能とは同時使用できないという制限が設けられており，ステルス機能を実行中，もし見つかって砲撃されればひとたまりもなく破壊されてしまう。

ステルス機能とシールド機能を臨機応変に使い分ける……という戦略も思いつくが隠密戦車はスロット数が最低の5個で，これをやってしまうと攻撃用兵器の搭載さえろくにできないことになってしまう。

一方，ステルス機能とシールド機能をこの戦車以外に実装しようとすると，それだけでほぼすべてのスロットを消費してしまい，これまたほとんど攻撃手段を持たない戦車になってしまう。

そしてステルス戦車はメインスクリーンおよび通常レーダーに対しては有効だが，オプション装備として用意されている対ステルス戦車用レーダー「位相差探知機」を実装すれば，その敵影をレーダーに投影することが可能だ。ステルス戦車はシールドを装備していないので，それこそ大雑把に砲撃してその数発でも当たれば破壊できる。さらに，衛星視覚装置を用いれば完全にその存在を見破ることすらできるのだ。

また，ステルス戦車は攻撃活動中はステルス機構が一瞬解除されてしまう特性を持っており，さらに被弾中も同様だ。よってチームメンバーの誰かが位相差探知機を実装して火砲であぶり出し，攻撃力に優れたチームメイトがそれに対し集中攻撃することで，いとも簡単に撃退が可能なのである。

「目にも見えない，レーダーにも映らない」一見無敵のステルス戦車だがTANARUSにおいてはこうしたバランス設定により，むしろ通常の戦車よりも「死と背中合わせ」の戦いを常に強いられていることになるのだ。

これまた「卑怯」呼ばわりするプレイヤーは単に自分が弱いか，あるいはチームメイトとの連携が下手……ということになる。

補給中の攻撃行動はずるい

逃げ出した相手を追いまわし，基地にたどり着くも，なおも砲撃を継続して相手を破壊する……というシチュエーションはよく起こる。これを「汚い」と呼ぶ輩がいる。また前線基地でダメージ回復中あるいは補給中に破壊されると「理不尽だ」という発言も出る。これはどうだろうか。

補給活動は次なる戦略活動に向けての下準備を行っていることにほかならない。すなわち，これを封じることは今後の戦況を考えればとても意味のある行動だ。もし，これができないとすると，戦車がその基地に居座ってしまった時点でその基地が奪えないことになってしまう。

また，補給中は装甲とシールドの回復が高速に行われるので一瞬にして破壊されることはまずない。もし，破壊されるようならばそれは自分がそれまでに「やられすぎた」ということだ。

そして，もし，補給中に攻撃を受けていることに気づいたならば，前線基地にいるのだから装備を変更交換することができるので，シールド能力を倍増する「防御壁強化機」を装備してしまえばまずやられることはない。

そして攻撃する側にしてみれば前線基地にいる相手を砲撃するということは非常に勇気がある行動なのである。というのは，本文で触れているように，相手の前線基地に近寄ると自機の電力が奪われてしまうというルールが採用されているため，攻撃を敵前線基地圏内で継続するということはいっそうの電力消費を促すことになる。万が一，ここで電力を使いきってしまい行動不能に陥ればまず破壊されることは必至だ。

もし，補給活動中に破壊されるようなことがあるとしたら，それは，その基地周辺に味方がいないということであり，自軍の連携が未熟だということになる。

補給中に遠距離射撃されることもあるが，単独のものならば，これは前述のとおり補給中はシールド＆装甲が高速に修復されるため防御壁強化装置を併用すれば破壊されることはまずない。複数機による遠距離砲撃の場合は……これは相手の連携プレイがうまいとしかいいようがない。しかし，すでに述べたとおり，遠距離砲撃にも弱点はあるため，結局これも「最善策」「必勝法」ということにはなりえない。

弱いヤツが破壊される

こうして別の視点でゲームバランスを見てみても，TANARUSでは装備や機種によって有利不利が発生しないような工夫がこらしてあることがよくわかる。

結局，「ずるい」「卑怯」と叫びながらやられていくプレイヤーはゲームシステムを理解しきれていない「弱者」にすぎないのである。

To Heart における感動の構図

古村 聡

ゲームというものはハマリやすい人もいれば，そうでない人もいる。多くの人がハマるゲームもあれば，当然そうでないものもある。ここではひとつのハマリパターンからゲームとプレイヤーの関係を考え分析してみよう。

いまさらなんでこのゲーム？　といわれそうな気もしますが。とにかく，To Heartです。このゲームはテレビアニメ化決定，冬にはPlayStationへの移植版が発売される予定の，とある高校を舞台にしたさわやか学園物恋愛系ゲームなのであります。システムとしてはアドベンチャーゲームに近い，と申しましょうか，別名「ヴィジュアルノベル」シリーズなんだそうで，プレイヤーに文章で読ませるタイプのゲームです。そして，昨年，パソコン版として発売され，Hゲームとしてだけでなく，パソコンゲームとして驚異的な売り上げを記録したゲームなんだそうであります。まぁ，この時期パソコンゲームがあまりにも不甲斐なかったのも確かなんですが(本当になぁ……ポンキッキのエデュテイメントソフトに負けてどうするよ，ホンマに)。それ以上にTo Heart旋風は凄かった。

くーっ，泣けるなあっ。マルチかぁいい，かわいすぎるっ。これに感動できないようなヤツは人間じゃないと断言できるっ！　だから，あなたも買え，さぁ，いますぐ買え。以上っ！

……なんて書くと，おいおい，お前はまたそれかい，ホレっぽいヤツだなって？　と古くからの読者の方にはいわれそうですが。違います，これは，Webページを「To Heart」and「よかった」あたりのキーワードを使って某検索エンジンでちょちょいとピックアップして，ぎゅっとまとめてみたものなんです。嘘だと思うなら，自分で調べてみてもらってもいいですよ。いやTo Heart関連のホームページのアツいこと，アツいこと。いまだに新作の「White Album」ページにならずにTo Heartで残っているページも実に多く驚かされます。

ちなみに，私は友人にこんなTo HeartなWebページのひとつを見せてみましたが。「うーん，こういう人とはちょっと友達になれないな」と正直に話してしまって5秒後にアスホールど真ん中にケリを入れられてしまいましたとさ。

さておき，それほど，To Heartというゲームは，ハマる人は劇的にハマってしまうゲームであるようです。しかし，どうしてそんなにたかがゲームに「ハマってしまう」のでありましょうか？その理由をズバっといいきれてしまうのなら，世の中には，それこそ大ヒット，大感動，のゲームばかりが世間に満ち溢れて，今頃みんなそれはそれは幸せになっているに違いないんですが(まぁ，それはさすがに無理です)。しかし，人がなにかにのめり込むにはなんらかの特徴的なポイントがあるはずです。では今回はきっちり，なんでこんなにマルチの話は泣けるのか，To Heartはいいっ！　といえるのか。一歩さがってプレイヤーの背後からしっかりここで考えてみることにいたしましょう。

なんでそんな面倒くさいことをするのかって？なぜならそれがOh! Xだから，なのであります。

あなたの目をあなたではなくすために

SCENE 1

♪Brand New Heart　いまここから始まる

(中略)

♪それなりの悩みも抱いて迷いも消えなくて

　この惑星(ほし)の上で

　　なにか求め探し続けて

*

セガカラにも曲が入っている，To Heartのオープニング曲「Brand New Heart」が流れて，オープニングデモが始まります。このオープニングでは，ゲーム中に使われているグラフィックを使っていくつかのグラフィックが表示されます。

このTo Heartというゲームは，とある高校を舞台にしたさわやか学園物恋愛系ゲーム。つまりプレイヤーの目的は「恋愛すること」なのであります。で，このオープニングでは，ゲームの舞台となる学校，校舎などで，走馬灯のように女の子たちがさまざまな表情を見せるのです。

これがまた「俺ぁ，高校時代なんて，周りにかわいい女の子なんていなかったしぃ(男子校だった，というのも可)，恋愛なんてなんかうまくいかなかったしぃ，すてでぃな子なんてできなかったし，クソクラエで，汗臭くて，弁当臭くて，殴ってやりたくなるような思い出しかねぇぞ，コンチクショゥ！」という私(でも，高校生活なんてそんなもんだよね)でも，なんだか甘酸っぱい青春に戻ったようで胸にジーンときちゃうんですよねぇ。

プログラム的には音楽にシンクロしてゲーム中のグラフィックを表示しているだけなんですが，が，ここでは，単にキャラクターたちの「グラフィック」を見せるダイジェストにならず，プレイヤーの心をここまで躍らせるにはいろいろなテクニックを使っています。そのなかのひとつが「このデモの冒頭には写真のグラフィックが現れる」ことです。キャラクターたちのグラフィックを表示するだけでもいいはずなのに，最初に現れるのは校庭の桜の木を背景にしたキャラクターたちの，ゲーム中では「これから」の高校生活が写っている「写真」なんですね。オープニングで，いきなり，女の子が次々登場する場面になってもいいのですが，この写真がオープニングのワンクッションになっているのです。

たいていのプレイヤーにとっては，(18歳以下禁止のゲームですから，タテマエを信じれば)もはや高校生活はすでに過ぎ去りし過去。写真の中，思い出の中のものでしかありません。いま，高校生活を思い出せ，といきなりいわれたら「あぁ，あんなこともあったねぇ」とせいぜい，いまの自分とは関係ない単なる思い出話としてのリアリティしか感じられなくなっている人が多いことでしょう。そこにいきなり，高校や女の子たちのグラフィックを表示しても，なかなかゲームに入り込めないのです。そこで，この写真が，ゲームにプレイヤーをのめりこませるためのワンクッションのタイミングをとっています。それから，写真のひとつの意味は暗示，です。写真とは「過去」を写したものです。写真を見て，過去に思いをはせる。そこからプレイヤーの心は思い出の，高校生活の世界に飛び込んでいくのです。

意図したものか，偶然そうなったのかはわかりませんが，このオープニングでの，この写真によるワンクッションのような細かい工夫がいろいろなところでなされているのが，このゲームの大きな強みであるように思います。

そして，ゲームに入ると，状況を説明するセリフが丁寧に描かれます。

*

SCENE 2

「風，また少し暖かくなったね」

　ゆるやかに吹く風を受け，髪を押さえながらあかりが言った。

　高台にある高校へと続く坂道。

　オレたちはまばゆい木漏れ日の下を通り抜ける。

　影から出た途端，まぶしい日差しが体を貫いた。

*

このTo Heart，ひとつのことを説明するのに，何度も，それも比較的長い文章で説明しています。筆者は，これに関してはゲーム中で少し削ってもいいのではないか，と思うところもあるのですが，このオープニングでは，文章が長い分，ゆっくりと時間をかけて，学校へ行く，いつもと変わらない朝，というシチュエーションに慣れさせていってくれます。

ともあれ，これであなたの心は高校時代を舞台にしたおとぎばなしのほのぼのした中にすっかりうずもれていくのです。

結末への道しるべ

さて，そんなわけで，プレイヤーはひとり，またひとりと女の子のラブストーリーを楽しんでいくわけですが。To Heartのとてもいいところは，この1つひとつのラブストーリーがとても完成度が高いってことなんですね。ストーリーがプレイヤーの心をぐっとつかんで離さないんです。で，主人公と女の子たちとのストーリーのなかでも，特によくできてて，とにかく感動的なのがメイドロボット「マルチ」とのお話なんですね。

＊

マルチは主人公の高校に8日間だけ試験のために研究所から送られてきた、試作型メイドロボットです。でも、ロボットのくせにおっちょこちょいで、かわいくてどうも憎めません。ロボットなんてただの電気仕掛けの機械じゃないか……と思っていたプレイヤーも、少しずつ、まるでマルチに心があるようにさえ思えてきました。

SCENE3

図書館で充電中のマルチに会う。主人公がマルチのようなアンドロイドも夢を見るのか聞いたところ，なんとマルチは「見る」という。それを聞いて主人公が，では，どんな夢を見ていたのか聞いてみた。

「たとえば、さっきはどんな夢を見ていたんだ？」
「さっきはですねぇ……」
「さっきは？」
「……」
「早く教えろよ」
「や，やっぱり内緒ですっ」
「なんだよ、それー」

＊

これはマルチのシナリオをエンディングへと導く重要なセリフ，マルチの夢についてです。いや，夢とは，寝ているときに見るから夢，ではなくて「マルチが望んでいること」ですね。マルチの夢については話題はゲーム中にこのあとも何度も出てきます。ゲームが進んでいくと，マルチの望んでいることは，

「誰かのためになりたい。あなたのそばで，あなたのために働いていたい」

ということである，というのがわかるんですね。メイドロボとして生まれたからには，そういうものかもしれません。そして，ゲームが進んでいくと，マルチの夢がかないそうな方向でという形でストーリーはエンディングへと突き進んでいくのです。

To Heartでは，キャラクターにもよりますが，シナリオ中にはキャラクター（と2人）の結末を予感させるようなセリフが次々に出てきます。このゲームでは，登場する女の子の数だけストーリーがあるわけですが，1人ひとりのシナリオは基本的には1本道です。その1本道の先へ先へと導くようにキャラクターのセリフや主人公のモノローグとして，しかも何度も何度も繰り返し，ときにはまるでバレバレに思えるようなエンディングを予想できてしまうようなヒントが出てきます。先ほど出てきたオープニングでさえよく見ていると，実はマルチや琴音になどは結末が想像できるような光景が現れています。

余談ではありますが，こうして，プレイしているうちに，プレイヤーは知らず知らずのうちに「この話はこうして，流れて終わるのかな」とプレイヤーの頭の中に結末のイメージをおぼろげに組み立てていきます。

ゲームというと，話の行きつく先，つまり結末を見せないように工夫する，というのが大原則のように思えますが，実はそうではありません。むしろ逆です。私の経験則ではゲームで，結果としてユーザーに「よかった」といわれるストーリーは，それがハッピーエンドであれ，悲劇的なものであれ，最終的な結末がこうなるんじゃないかという予想が，結局そのとおりになったものがかなりの数を占めています。

ただ，ただ筋書きがオーソドックスなもので，結末がプレイヤーの思う方向にさえ進めばいいシナリオになる，というわけはありません。いくらいい結末があってもそれにいたる過程を話を練り上げていかなければいいシナリオにはならないのです。

アンビバレンス

いくら幸せな結末があっても，ただ一本調子にそれが実現できてしまえば，それはそれでもの足りなく思います。贅沢ですが，それが人間，というものなんですね。特に，To Heartの場合は，基本的にプレイを進めていく動機は「Hシーンが見たい」「それも早く全員のを見たい」でしょうから，気持ちがはやって早解きをしてしまい，結局ストーリーが心になにも残せなかった，ということになってしまうことも十分考えられます。

しかし，はやる気持ちを抑えさせて，ゲームのストーリーに目を向けさせなければ，結局エンディング確認作業だけのゲームになってしまいます。物語に対するイメージがとても薄いものになってしまい，結局人の心に残らないのです。ここらへんがシナリオを重視したゲームの難しいところでしょう。

そこで必要なのが，「アンハッピーな状態」です。To Heartでよく使われているのは，シナリオ中に出てきたエンディングのヒントとはまったく逆の方向をキャラクターの口を通じていわせることで，プレイヤーに「ここで自分がなんとかしてあげられたらいいのに」と思わせることにあります。自分の前にいる女の子をなんとかしてあげたい，今，ここで抱きしめてあげられたら。いや，それよりあのときああしていれば……。

そして，それぞれのシナリオについて，プレイが順調にいっているように思えているときに，突き落とすという作業が必要になるんです。これが，To Heartをプレイ後に心に深く突き刺さるものにしている，テクニックなのです。お話には「どんでん返し」が待っています。

＊

「わたし、そろそろ帰らなければなりません…」

＊

マルチは，研究所に帰らなければならないと，まるでシンデレラかかぐや姫を思わせるセリフを吐きます。そうです，ストーリーは，せっかく作

り上げた2人の関係も結ばれないで終わろうという方向に転げ落ちていくのです。マルチは，衝撃的なことを告白します。マルチはこのテスト期間が終われば，データが量産機のために使われ，彼女自身の記憶や心は研究所の暗い倉庫の隅で永遠に眠ることになる。つまり，帰ったらもう二度と主人公とも会うことはできない，というのです。

実はマルチは，マルチの心の中でも2つの相対する心がぶつかっています。マルチの夢は「誰かのためにお役に立ちたい」ことなんですが，

「主人公のために残りたい」

「生まれてくる妹（量産機）のために，自分を使わせてあげなくてはいけない」

と同じ相対する「お役に立つ」ことの板挟みになってしまうのですね。そして，その夜，主人公と結ばれたマルチは，翌朝，研究所に戻ります。

＊

SCENE5

「どうも，お世話になりました」

「……おい，いいのか？　本当に帰るのか？」

マルチを引き止めるようにプレイヤーはいいます。

「……マルチ，お前，そんなんでいいのかよ？　さびしくないのかよ？　……怖くねーのかよ？」

「わたし，ロボットですから」

「ロボットだから，平気なんです。……ロボットには，もともと心がありませんから。……寂しい気持ちも怖い気持ちもなくて，あるのはデータだけなんです」

「このまま研究所に帰ったら，お前はもう……」

マルチはほんの一瞬，表情にかげりを見せたが，すぐにまた笑顔に戻ると，

「はい，帰ります」

はっきりとそう答えた……。

＊

ストーリーはプレイヤーの思いとは関係なく，2人を引き裂きます。プレイヤーは「ここで自分がなんとかしてあげられたら……」と思わされてしまいます。しかし，これまで，自分の思うように女の子との出会いを作り出して，ここまでプレイも順調に進んでいたのに，ここではプレイヤーの介在する余地がほとんどありません。もちろん，「なんとかしたかった」「なんとかしたい」と思っても，もちろん，戻ったところでどうにもできませんし，これから先それができるかどうかはシナリオ次第です。ストーリーはあらかじめ作られたシナリオに沿って先へ先へと進んでいってしまいます。青春をやり直しているのに，また青春に苦い思い出を積み重ねてしまう……。自分が運命を自在に操れる力を持っているような錯覚にとらわれていた，プレイヤーを無力感の嵐に叩き落とすのです。

しかも，この「どんでん返し」では，先ほどプレイヤーが抱いていた妄想をも利用します。ストーリーが順調に流れているときに，ハッピーエンドへの妄想を描いていたように，プレイヤーはここでもエンディングがどうなるのか，と思ってしまいます。もしかしてこのまま「悲しい結末」を迎えるのか？「ハッピーエンド」なのか「悲しい結末」なのか。プレイヤーは戸惑います。しかも，シナリオはこのまま悲しい結末の予感を強めるように働いていきます（この辺，ホントにうまいんだよねー，このシナリオ。シナリオライターのドラマツルギーを感じます）。

＊

そして月日は流れ去り……プレイヤーは大学に進学し，努力の甲斐あって，なんとか発売された量産型のマルチを手に入れた。しかし，量産型のマルチには試作型と比較して大きな変更が加えられていた。それはコストダウンのために，「心を一切持たない」メイドロボになっていたことだった……。呼びかけても呼びかけても，マルチがあの笑顔を見せることはなかった。永遠にマルチは帰ってこない……。

＊

もし，ほかのキャラクターでエンディングを迎えた人ならわかるように，この辺はゲーム中の背景＋キャラクターのパターンのグラフィックでなく，フル書き込みのグラフィックが続きます。そう，明らかに「そろそろエンディング」のパターンなのです。甘い「2人だけ」の関係を作るための，やり直しの高校生活（＝このゲーム）。それなのに，君を永遠に失ったまま，終わってしまうのか……。ここでプレイヤーの「なんとかしたかった」という後悔の念は最高潮に達するはずです。感情移入の高まったプレイヤーなら，2人の悲しい運命にここで涙が出てしまう人もいるかもしれません。しかし，お話はどんでん返しのまま終わってはいけません。結末はやっぱり，聞き手の望む方向へ。それがストーリーテリングの王道なのです。

踊らされたいあなた

さて，こういうことをいうと「それってストーリーに引っぱられて，踊らされてるだけじゃん」という人がいるかもしれません。この疑問には「まったくそのとおりです」と答えてあげましょう。

プレイヤーは実はゲームに踊らされたい，と心の底で思っていることが多いのです。たとえば，ゲームではなくてもいいです。映画を見ていて「踊らされんぞぉ」と身構えて見る人はいるでしょうか。「タイタニック」を見るのに「俺はなにがあっても悲しいなんて思ってやらないぞ～。誰が死のうが知るもんか～」なんて，心を鬼にして劇場に向かう人がいるでしょうかね？　それはどう考えたって逆ですよね。見る側は，タイタニックとともに沈まなければならない，デカプリオを「かわいそ～」と思うために見に行っているんですよね。

映画にしても，ゲームにしても，人というのは感動したり，涙を流すタイミング，というのを自分で計っていたりするものなのです。

幸せな結末は，マルチとの関係の場合，こんなふうに訪れます。

＊

そんなある日，俺あてに宅配便が届いた。

差出人はマルチを作った研究所からだった。中身はソフトウエアだった。説明書のとおり届いたソフトをマルチにインストールする，すると……。

「……」

「……マルチ？」

「マルチ！」

今度は強く叫んだ。すると……，

「……浩之さん……」

「マルチ……」

「……あ，会えた……また，会えました……」

広げた腕に勢いよく飛び込んできたマルチを，

オレは思いっきり強く抱きしめた。
「マルチ，マルチっ!」
「……わたし，…てっきりもう会えないんだとばかり思ってました……」
　マルチはぎゅっとオレにしがみついた。
「今日からはずっといっしょだぜ。なんたって，オレは正式にお前のご主人様になったんだからな…」

＊

　とマルチにはマルチの心が戻って，プレイヤーとマルチの運命はここにふたたび結ばれ，幕。誰もが涙々，のハッピーエンドが現れるのであります。

　最終的には結末は途中でプレイヤーが思っていたとおり「ボクとあの娘で結ばれて，完了」なわけですが，なんだかんだと，プレイヤーを「まさかこのまま終わりってことはないよねぇ」とドキドキさせたりを繰り返させてプレイヤーの心を揺さぶる。悲しませたりもしながら，持ってくることで，お話をとても劇的にできるのです。ありきたりかもしれないけど，マルチのストーリーが感動できちゃう理由はこういうことをきっちりストーリーとしてやっているからなんですね。

　ここは，素直に「ああっ，踊らされてる～」なんてうめきつつも，ついに主人公との再会を果たした感動の場面に，素直に泣く。これが，プレイヤーの清く正しい姿なのであります。端から見て美しいかどうかはともかく。

　このゲームの場合，それはそれはもぅマルチに燃え萌えっすよ～，という強烈なファンが多い半面，
「買ってはみたけど，まだ全然手をつけてない」
という人もときどき聞きます。曰く，
「文字が多くて読むのがつらい」
ということなんだとか。確かに，画面に表示される文字数は横に23文字，平均9行程度，平均1画面で原稿用紙1枚分，電車の中なら車内吊り広告1枚分くらいの文章を読むことになるわけです。画面で文字を読み慣れていない人は文章の字面だけを追うことになってしまうかもしれません。

　しかし，ゲームの面白さ，特にこのTo Heartではまずはこのゲームの持つ世界にどっぷりハマり，キャラクターと自分との関係を文章から受け取り，感情移入して，盛り上がっていかないとわかりにくい性質のものです。それなりに，受け手に読解力，つまり読んでそれを自分の中でちゃんと消化して自分の中のイメージにする力が必要なのです。イメージする以前のところでつまずいてしまうと苦しいでしょうね。逆に，このゲームで感動できる人は，それなりの読解力と感受性をちゃんと備えられた人なんだ，って誇ってもいいと思います。

＊

　さて，今回の原稿は，古くからのOh!Xの読者の方なら「なんか同じようなことをどこかで読んだことがあるな，デジャビュかな？」などと思っている方もいるかもしれません。それは偶然ではありません。正直に白状すると，実はこの記事は1988年4月号の「不思議の国のリアリズム」，1988年10月号「イースのゲームデザインを読む」という斎藤晋氏の記事の構成を参考に，To Heartに当てはめてみたものなのです。10年前にいわれたことがほとんどいまでも通用するのですから，驚きといえば驚きです。そして，さらにさかのぼると1987年10月号（あ，この頃はさらにOh!Xの前身のOh!MZだ）の特集「GameDesignを考える」にこんなことが書かれていました。
「そして当然のことながら，ゲームは制作者だけによって作られるものではない。ゲームの面白さはプレイヤーによっても左右される。そもそも，ゲームを楽しいと思えるには，楽しいと思えるだけの知性が必要だ。それを相手の知性を知る能力である，といってもよい」

　まったくもって，いまのゲームにもそのまま当てはまる言葉であるといえましょう。面白いゲームの基礎っていうのは，実は何年たっても，変わらないものなのかもしれないですね。

column　画面消去のテクノロジ

　面白さ，ということに関してはあまり変わっていないけど，この10年で大きく変わったマシン環境。ただ，画面効果あたりのテクニックって，かなり煮詰まってきてしまったのか，それともWindows環境への対応でそこまで手が回らなかったのかわかりませんが，ここ数年もうあんまり進歩がないような気がします。確かにフルカラーでアニメを描けるようになったし，目パチ口パクも手間さえかければ24fpsのフル画面アニメーションだってできる。画面消去も縦に筋状に消す，点状に消す，フェードイン/アウトもできるので，それなりのことはできるんですが……，やっぱり少しはこの辺ももう少しテクニックが出てきてほしい気がします。

　そんななかで，私がほしいと思うのは「オーバーラップ」。これは，テレビアニメなどでは頻繁に使われている技法のひとつで，その名のとおり，以前のシーンをフェードアウト，これから現れるシーンをフェードインさせながら画面を重ねあわせて表示して自然にシーン転換などを行う方法です。

　これがあると，ストーリー的につながりのあるシーンのつなぎなどに使っていれば，かなりプレイヤーの受ける印象が違うと思うんですがねぇ。256色モードではまず無理，フルカラーでも色を何度も計算しなければならないのでまっとうにやるとCPUパワーを必要とするなど，ちょっとプログラム的に面倒だ，というのはわからないのでもないんですが，このMPEGだってソフトで再生できる文明開花の御時世。やればできるはずなんで，これからのゲームには使ってほしいなぁ，と思う今日このごろなのであります。

魂に響くゲームたち

横内威至

久しぶりのレビューっす

Oh!X の皆さん，久しぶり。X68000は少ないながらも良質なゲームに恵まれてたよな，確か。いまはどうだい？　いろんなゲームを楽しんでるか？　コンシューマゲームも，ハードの進化にあわせて，たくさん出ちまった。これではゲーマーとしては大変だ。全部になんか手を出せない。

ゲームを選ぶとき困るよな。かといって，雑誌のレビューじゃ偏見だとか政治的な理由だとか，大人の世界の話でよ，なんか，こう，しっくりこないよな。人によって好みもあるしさ。レビュアーの人間そのものを理解しないと，参考にしづらいってのが本音かな。

よって，久しぶりに俺もレビューしてみようと思う。淡々と，赤裸々に，そして高飛車に。当然，俺なりの偏見でレビューする。しかたないじゃん。俺が書いてるんだから。点数とかはつけない。よい作品だけを紹介する。少なくとも，飯を何食かはカップラーメンで済ましたり，来年には捨てちゃうTシャツを1，2枚程度諦めたりしてでもプレイする価値はある。そんな程度の倹約でこれほどの体験ができりゃ，それは非常にいい選択ってことになるぜ。ソフトの価値について必要以上にハッスルしちゃう人がいるけど，もうちょい冷静になってみよう。

蒼天の白き神の座
アーク・エンターテインメント

人として，やるべきゲームのひとつだ。なぜなら正しいヒト科の遺伝子は，自然に憧れるようにしっかりとプログラムされてるからだ。

久しぶりに，ゲームの本質という部分で素直に評価できる，最近では珍しいストイックな内容である。内容は，純粋な登山シミュレーションゲーム。戦争でもないし，経営でもないし，さまざまな外部要因（世界に間接的に影響力を持つもの）を気にせずにシミュレートできる内容であるだけに，世界観やシステムの整合性，完成度は抜群だ。ゲームだからできてしまうというような，とても現実ではありえない戦略は通用しない。リアルなシミュレーションだ。

ゲームで頭を使いたくない人，中間管理職なんかで人間管理に嫌気がさしてる人，自然とか冒険とかに少しも心が躍らない人には向かない。

やってみるまでどういうゲームになるかわからないので，日記のように体験記を書いてみる。あらゆる行動に危険がつきまとうので，なるべく安全な“戦略”を練ることがこのゲームの面白味であり，体験記を書かないと全貌が伝わらない。単純に「こうすればいい」というプレイ方法はないのだ。こういうやり方でどうする，ってのもやはりないのだ。状況にあわせて臨機応変に対処するゲームなので，その代表的な状況，そして原因を連ねることにする。しばらく真面目に，淡々と書こう。ちなみに日記をつけた経験は過去一度もない。その手ので，続いているものはパチンコの収支表だけだ。

●出発準備

まずは登山隊の編成だ。初期段階で数人登録されている。若手は重要だ。みんな50歳ぐらいで引退してしまう。よって，初期メンバーで可愛がるのは藤田桂子24歳。女性隊員は少ないからヒイキしてしまう。同時に，今年も新規隊員を募集。若手ならば技術レベルは問わずにゲットする。1回登山するだけで相当レベルアップするからだ。

次いで装備の確認。とりあえずはおまかせコースでルート工作セット，テント，35日分の食料なんかを用意して登山の準備を整える。まずはもっとも低いマヌーツェ。複数ルートの近接するカマプルBC（ベースキャンプ）から攻略。季節はもっとも穏やかな6月。素人隊員のレベルアップがなによりの目的だ。

●登山開始

今回のパーティは21人。Sクラスの精鋭から，Eクラスの素人隊員までさまざまな顔ぶれ，というとベンツみたいで高級だ。まずは4つの班に編成する。

1班：エリートコース。特攻隊だ
2班：まあまあ特攻隊
3班：基本的に運搬部隊
4班：観光旅行部隊

下のレベルにはそれなりのリーダーをつけて，無理のない程度に行動してもらう予定だ。

朝5時。まずは1班にはBC付近（4581m）のルートを探してもらう。その他は休息。とりあえずみつかったルートの先まで歩いてもらって，少しでも高い位置まで探検してもらう。

いきなり2方向へのルートを発見。まだまだ安全な地形。とりあえず，中央を突破したい。左は大きく迂回しそうで，距離が長いため避けたいところだ。よってとりあえず中央に向かうルートをさらに探索する。午後2時，5500m付近に雪崩，落石の危険性の少ないキャンプ向けの場所を発見し，今日は終了。午後6時，BCで飯食ってゆっくり休憩だ。

●高度順応とキャンプ

6月2日　快晴。昨日見つけたキャンプ予定地まではとりあえずルート決定だ。1日あれば往復できる距離だ，多分。この移動距離と難易度のサジ加減が微妙だ。探索にしてもそうだが，とにかく夕方にはキャンプに戻れるようにしておかねばならない。ビバークは凍傷などの危険性が高く，ここぞというときの突貫作業で使うにとどめたい。

今日はさらにその先の探索，そして第1キャンプ（C1，5574m）までのルート工作を行う。まず1班はC1から先のルート探索。2班，物資の運搬。3班，ルート工作。4班，キャンプ設営。レベルの低い班には安全な範囲でいろいろと経験してもらう。荷物は安全なうちは無理して多めに運んでおく。まっ先にキャンプで宿泊可能なように，食料だけは多めに運ぼう。とはいえ，ある程度先を見越し，多すぎても困る。下山するときにゴミを残してはならないのだ。あくまで物資は適量にとどめておく。

偵察部隊がさらに先のルートを発見。やや危険性の高いルートなんで，別ルートも探索。午後2時。キャンプ設営も終了し，荷物の運搬も終了した。偵察部隊は先のルートを発見したが，ミスった！　順応していないはるか上の高さでの活動が続いたため，高度障害を起こしてしまった。数人は軽い凍傷にかかり，急いで休息させねばならない。無謀にも，順応高度より1500mも高い地点で作業し続けてしまったのだ。

高い場所では酸素量が少ない。高度4000mでは通常の2/3ほどだ。順応しないと体の末端まで酸素が供給できない。つまり，凍傷を引き起こす。しかし，人間もタフなもので，ある程度の負荷をかけたあとに休息をとることによってより少ない酸素でも行動できるようになる。筋肉の成長と同じだ。このゲームもまた然り。高いところほど，作業は慎重にやらねばならない。疲労も激しく，すぐに行動力や気力が低下し，ときにはキャンプへ帰還することも苦しくなってしまう。常にキャンプへ帰還するマージンを残して作業しなければならない。午後7時，少々遅れて全員BC帰

還。焦ってはいけない。まだ先は長い。

●ルート選択

6月3日　午前2時。3班の船岡隊員が体調不良を訴える。しばらくはBCに残ってもらおう。午前5時，起床。天気は下り坂，風が強い。1班は高度障害が長引いたせいで全然順応できてない。今日は2～4班により高度まで進撃してもらい，BCで休息，とりあえずC1までの順応をすませてもらいたい。まずは2班，C1先の地帯の探索。3，4班，物資の運搬。午後12時，2班探索中。C1先の壁を避けたいため，ほかのルートを探ってもらう。

ルートにはさまざまな地形がある。雪田，氷壁，岩壁，稜線のリッジなどだ。地形によっては雪崩，落石の危険性が高い。壁であれば技術なしには滑落してしまうし，稜線では強風が吹きやすく，時には身動きがとれなくなってしまう。左右崖の細いエッジ上で，風速30ｍなんて状況もザラだ。極力安全なルートを探しておきたい。避けられなければ最低でもルート工作をし，危険性を抑えておかねばならない。壁の中腹で身動きがとれず，食料もなしにビバークしなければならない状況ではグイグイ体力を失ってしまう。レベルの低い隊員には無理を強いれない。簡単に立ち往生してしまう。

午後3時，困ったことに風は強くなる一方。2班はC1先で強風につかまってしまい，行動速度が低下，3班もルート途中で身動きがとれない状況だ。2班はまだC1にも順応できてないので，無理にでもBCまで帰還してもらう。午後9時，2，3班はルート途中でビバークするはめに。すまん。

●アクシデント

6月10日　曇り。気圧が低下している。午後には雪が降り始めた。風も強まっている。そんななか，午後7時にC1からC2予定地へ向けてルートを工作中の2班が途中の壁で滑落！　あとわずかで壁の工作が終了だったので無理してしまった。しかも，ルートを見失い，何人かは大怪我，そして気絶している！　タイミング悪いことにC1にいた4班は高度障害，助けにはいけない。強風のため，BCから応援を出すも途中で立ち往生してしまっている。とりあえずはビバークで朝まで待つ。が，

6月11日　午前2時，気絶していた隊員のひとりが息を引き取った……。午前5時，捜索開始。2班は連絡だけはつくものの，体力も低下，移動不可能。午前8時，2班発見！早速合流し，動けない隊員を1班の隊員が運ぶ。なんとかBCまで持ってくれ，と願うが，疲れきった2班は途中で再び滑落，怪我を負っていた隊員をまた1名失ってしまう。午後4時，BC到着。数人が病院に送られ，死んだ隊員は日本へ送られる……。すまん。隊長失格だ。隊を再編成，合計3班に仕切り直す。

6月13日　C2設営。1班は探索活動のため，すでにここまでは順応済み。C2で宿泊，明日はいよいよ頂上までのルートを探索する。

6月14日　C2から頂上までのルートを発見！　途中，スノーブリッジ，そして最後は稜線のナイフリッジ(切り立つ崖)だ。風のない日にアタックすべく，各隊はC2までの順応を済ませる準備を整える。頂上付近の高いところはより危険だ。酸素が少ないこと，そして地形自体，より急な斜面であったり，壁であったり，鋭い刃物のような尾根であったりする。大部分はモロに風の影響を受けやすい地帯になってしまい，タイミングを誤ると身動きもとれない状況となってしまう。頂上付近ではアクシデントは致命的だが，もっともアクシデントが起こりやすいのは当然頂上付近なのである。

●アタック

6月15日　午後2時，3班，アタック開始！　やや風があるが，絶好のチャンス。C2からならば距離もなく，素人軍団でもアタック可能だろう。食料だけ持たせて出陣。スノーリッジ，風速18ｍ。滑落だけはごめんだ。頂上付近での遭難は隊の全滅につながる。救出部隊の派遣が容易でなく，しかもリスクも高く時間もかなりかかってしまう。

午後8時，闇夜の頂上に立つ！　成功だ。この瞬間のために，絡みきったさまざまな状況をかろうじて繋ぎ止め，戦略を練ってきたわけだ。もう少し長引けば混乱の一途をたどっていたかもしれない。

この解放感に快感を覚えるのだ。斜面がきついために予想以上に進行が遅れたが，まだ十分に体力もあり，問題なく帰還できるはずだ。アクシデントだけはごめんだ。天気が悪くなる前にC2まで帰還。少々無理があるが，そこまで厳しい山ではない。ときには大胆に行動する必要もある。

6月17日　曇り。気象情報によれば天気は回復傾向。C2で待機していた1班に，午後1時にアタック命令。午後5時，登頂成功。天気はすっかり晴れ，360度のパノラマが広がる。あとは2班のみ。

6月18日　2班登頂成功。帰りに滑落にあうが，運よく無事だった。ここで事態はようやく楽なほうへ収束していく。

●撤収

全員のアタックに成功した。あとは撤収。キャンプを片付け，余剰物資もすべてBCまで持ち帰る。環境に優しい行動をとらないと環境委員会からクレームがついてしまい，隊長の経歴にキズがつくのだ。各自分担し，C1，C2を撤去，BCに帰還。作戦は終了だ。犠牲は大きかったが……。

ゲーム的なシステム部分の話

隊員は登頂に成功したり，困難なルートだとか条件によっては組織からアワードが贈られる。すげえ業績を積めばハッタリのきいた冒険家として名を残す人生を歩ませられるわけだ。

プレイヤー自身は作戦の成功によってランク付けされる。ルート開拓の総距離，日数，物資の量，登頂成功隊員数などによってだ。長い年月をかけて，部隊を強化し，最高峰Ｋ０を極めるのが使命だ。ちなみに，山は5つある。それぞれにベースキャンプが5つ程度あり，それぞれにさまざまなルートが用意されている。

その他のデキ

画面構成など，堅く丁寧な作りで，センスもよい。少々，ポップダウンメニューの選択がわずらわしい部分もあるが格闘ゲームのコマンド入力のように体が覚える。ムービーのデキ，そのはさみ方，いい仕事がしてある。わざとらしい見せ方はなく，ドキュメンタリー的な映像が格調高い。

けっこう淡々としたイベントも，余計な情報がなくてちょうどいい。想像力によって装飾してもいいし，単純にゲームを楽しむにもピッタリだ。愛を持って育てた隊員が死んだりすると本当に切ない。

要求ももちろんある。それはあたかもそこに立っているかのようなビジュアルがほしいのだ。きつい壁とはいえ，やはり記号的であり，見るからに厳しそうな映像はないのだ。ここは細かい部分までモデリングし，ぜひ，隊員の目に映る映像そのものを我々にも見せてほしいところだ。次回作があるのなら，この部分を派手に追求してみてほしい。現行機で難しいのはよくわかるのだが。

とりあえず，やれ。来年には捨てちまうシャツ1枚我慢して，パッキー1枚我慢して，買え。

天誅
ソニー・ミュージックエンタテインメント

別にソニー系列をひいきしているわけではないが，2本目は天誅(あとから知ったけど浦川さん関連の作品ですねえ，これ。私信っすが，忙しくて連絡とれなくてごめんなさい，今後ともよろしく)。

みんなTomb Raiderはプレイしたか？　これは和製のソレと思って問題ない。違うのはダイナミックにアクションしてしまって全然構わない点。ある意味懐かしのストライダー飛竜に通じるような，適当に快くプレイすることのできるアクションだ。かといって大味って意味ではない。正しいプレイは忍びに徹すること。どうせみんなプレイしただろうけど，メタルギアと一緒なわけだ。

臭いもいい。なんたってモーションはショー・コスギ，ケイン・コスギ。忍者を生業とする，宇宙最大にして最後のヒーローだ。どの角度から見てもこのソフトは本物。「B級大作」という言葉に過敏な人はもはや「天誅」のトリコだ。悪い意味ではないぜ。あくまで「B級大作」モノにのみ許された，スルメのようなテイストってこと。独特の毒，味，珍味，クセにはまるとやめられない世界

なわけだ。我々の知ってる忍者はNINJAであって，SHADOW DANCERなんだよ。このアメリカ仕立ての忍者こそがヒーロー性ある忍者なのだ。これこそB級テイストであり，鋭い刃物となるわけだな。あくまで俺は褒めてるぞ，この味を。

NINJAと時代劇の核融合

プレイヤーは2人。力丸25歳，彩女(アヤメ)21歳。熱血漢と肝の据わったイイ女。正しい設定だ。カッコイイめの主人公2人を作れといわれれば，俺もこう選択する。ヒーローの基本だ。ちなみに，アヤメでのプレイのほうがなぜか難しい。

アクションは当然，走る，しゃがむ，跳ぶ，斬る！ 忍びゆえ，張り付く，忍具を使う，ことが可能だ。状況や組み合わせによってさまざまなアクションが可能だ。三角飛びもちゃんとあるから安心だ。

忍具は多数ある。手裏剣は当然として，アイテム，数々の忍術，そしてこのゲームのダイナミックさを演出する鉤縄だ。遠方に楔付きのロープを打ち込み，ダイナミックに移動できるカッコイイ忍者アイテム。これさえあれば城だって簡単に登れるわけだ。自分より高い場所にしか打ち込めない。こいつのおかげで，NINJAらしいアクロバティックな行動が可能になるのだ。

さあ，細かいことはいい。任務だ。まずは……。

「悪徳商人を成敗せよ」

いいじゃないか！　まさに望んでいた世界だ。悪徳といえば越後屋と決まってる。「成敗」，これこそ正義のヒーローのみに許された崇高な活動だ。変態仮面を覚えているか？　彼も由緒あるヒーローだな，つまり。

殺しの美学

越後屋の豪邸。多くの住居に広い庭。当然，悪人だから見張りや犬で武装している。望むところだ。潜入しろといわんばかり。忍びたるもの，慎重に潜入しなくてはならない。見張りに見つかるなんて失格だ。襲われたことにさえ気づけないほど迅速な殺しを心がけるのだ。プロの仕事は完璧でなければならない。

さっそく鉤縄にて，外壁に上る。正門から入る馬鹿はいない。広大な庭を避け，倉の裏に回る。下では無能な見張りがルーチンワークでうろついている。本来，なにごともなかったかのように仕事を片づけるべきだが，やっぱりオカズもほしい。手下どもも成敗してくれようじゃないか。うろついている見張りは，基本的には馬鹿だ。しかたない。ボスは女郎とお楽しみ中だってのに，そう滅多に現れない刺客を待ってなければならないのだから。

歩き出した瞬間，背後に音もなく忍び寄る。いまだ。しっかりと背後をとり，警戒されなければクリティカルな殺しが行える。背後から片腕でホールドし，一瞬にしてノドを斬るのだ。キメのポーズとともに，血の雨が降る。これこそ100点，いい仕事だ。もちろん，無抵抗な町民なんかを殺めてはいけない。正義の鉄槌は悪のためにあるのだ。

そのまま角を偵察。壁を背に，あたりを見回す。遠くに手下がいる。忍び寄るには間がありすぎる。手裏剣の出番だ。狙いをしっかりと定めて撃つ。一撃では仕留められない。が，自分は気づかれていない。そのまま数個の手裏剣で地獄行き。決して存在を悟られてはならない。はずした手裏剣は回収する。ヒーローとはいえ，庶民的な価値観を忘れてはいけない。日本の政治家がヒーローになれない致命的な部分は，まさにここにあるのだ。

成敗

幾多の屍を築いたあと，ついに越後屋を発見。某ラグビー部よりストレートな表現で，女郎と遊んでいる最中だ。「悪行の数々，見逃すわけにはいかん」，用意したセリフと共に成敗開始。と，用心棒，浪人の佐々木半兵衛登場。重要なシチュエーション，これぞ日本。お決まりのセリフで挨拶を交わし，死合開始。越後屋は鉄砲攻撃，さすが悪党。

ところが，こいつらも真正の間抜けだ。越後屋との間に半兵衛がいても構わず発砲してくる。おかげで半兵衛はなにも仕事しないうちに越後屋が成敗してくれてしまうのだ。素晴らしい。懐かしきピットファイターの臭いがする。俺の大好きな設定だ。意図したのかせずにかはわからないが，こういう間抜けな状況はもっともうれしい。越後屋自身はただの肉塊。助っ人なしではイチコロだ。「わしの金があぁぁ」，悪徳商人には不可欠な遺言がやっぱりうれしい。

忍術

任務が終了すると，必殺，斬殺，隠密，非道の採点からランク付けされる。敵に見つかることがなによりも減点されてしまうので注意。あくまで忍び，闇の活動なのだ。見事免許皆伝となると，新たな忍術を授かる。これがまたシブい。動物の鳴きまねで敵を欺いたり，変わり身の術なんかは本当にダッチワイフ程度の人形をダミーにしてしまう。

忍具もいい味だ。痺れ団子，毒団子であるが，「あまりにうまそうなので人間も引き寄せられる」らしい。この絶妙な間抜けさはこのゲームの中でも派手な隠し味だ。

その他，「邪教卍教の御神体」「南蛮海賊の来襲」「密書を運べ」などのおっ勃ちミッションの数々。キ○ガイとしか思えない邪教の連中も，本気で吐き気をもよおすほど，近寄りがたい言動をとってくれる。天狗や鬼，正しく間違った日本が舞台だ。

ステージも多彩だ。城下町，城，崖，緻密な行動，ダイナミックなアクション，フルに駆使しなければならない。

最後に

本当はもう1本，いろいろな部分で冒険していた「FRONT MISSION ALTERNATIVE」も取り上げてみたかった。ちょっとスペース足りなかったかもしれない。少しでも紹介したゲームの面白さが伝われば，と思っている。久しぶりの文章で，至らぬ点も多いかと(もともとそうだったかもしれない)。あんまり大人になってない証拠だ。

時代は変わって，ゲーム開発は本質の部分での勝負ではなくなってきてしまった。ムービー先行型とか，ある意味，かつてのゲームは終わってしまったのかもしれない。メジャーになるというのはそういう意味だったのだろうか。それも重要なことですが，チンケなドットの中でも楽しみを見つけてたあのころが懐かしい。年を経たせいなのか，それともゲームのネタは出尽くしてしまったのか，それとも技術戦争の急速化だけでいまは手いっぱいなのか。大人になったから悩んでいる。

とりあえず，今回紹介した2本はゲームの本質部分で勝負できるてるイケなソフトだ。ぜひプレイしてみてほしい。

Emi's Real Virtual Girl's World

シャネルの深淵

野沢絵美

こんにちは，野沢絵美です。ソフトハウスの広報宣伝部で製品広告とカタログ制作，CGデザイン，ホームページの作成をやっています。これは会社でのお話で，自宅ではテレビゲーム，マンガ描き，海外旅行，おしゃれに燃えているOLです。おしゃれのなかでもお気に入りのブランドがシャネルです。

ときたま，永野護さんのマンガ『ファイブスター物語』にシャネルのロゴがさりげなく登場すると，

「わい，アイシャ様はシャネルのコンパクトをご愛用だ」

と，ついうれしくなってチェックしちゃいます。永野さんってマンガで使用する衣装や小物を既存のブランドをもとにデザインし，登場人物の演出に使っちゃうのが上手です。『ファイブスター物語』は男性だけでなく，おしゃれ好きな女性に受けています，……というか少なくても，私ともうひとりのおしゃれ好き女性には受けまくっています。

私のこだわりっていうか，マンガやゲームを選ぶ基準のひとつに『登場人物たちがおしゃれ』ってのがあります。これは，単に着飾っていればいいってことではなく，服装とかアクセサリーって世界観や時代を表す重要な小道具だから，ファッションがきちんとしていれば，さらにそれが綺麗なら内容もしっかりしてそうだな～と，ついつい買ってしまうのです。

なので，この本を読んでひそかにゲームデザイナーを目指している方はおしゃれってのを意識しても損はないのでは？ とも思ってます。キャラクターのイラストとか画面がイマイチで雑誌の評価記事を読み飛ばされちゃったり，パッケージを手にとってもらえないと，せっかく作った斬新なシステムも超感激ストーリーもたくさんの人の目に触れないで終わっちゃいますしね。でも，

「とはいっても，どこから取りかかったらいいのかわからない……」

とご担当編集者（U）氏がぼやいておりました。確かに女性の私ですら山のようにある女性誌から自分がほしい情報を探しだすのは至難の技。とのことで，今回は私がシャネルファンになったことの話をします。メジャーなブランドですし，ちょっとはファッションの勉強の参考になるかもしれませんよ。

シャネルを買おう

高級ブランド買おうかなと思いついたのが2年ほど前でした。毎月LDを買っていたOVAの連載が終了し支出が減ったので，なんとなくお金がたまるようになったからです。

「なんに使おうかな～，メジャーな海外は行っちゃったし～，パソコンはあるし～，DQ6やっちゃったあとだからほかのRPGゲームなんかやる気がしないし……」

と悩んでいたときに，いつも楽しみに見ているテレビ番組の『ファッション通信』が目に入ったのです。放映していたのはパリコレ[*1]のオートクチュール[*2]のショウ。考えてみると，私は，おしゃれは好きだけど，高級ブランドには手をだしたことがありません。

「そうだ，シャネルを集めてみよう」

と急にシャネルがほしくなり，週末に日本橋三越のシャネルショップへ行ってみることにしました。

なぜ，いきなりシャネルなのかというと，いちばん有名だからです。そうです，私はミーハーなんです。

シャネルを集めるといっても，シャネル製品には服，アクセサリー，バッグ，化粧品といろいろな種類があります。一般的に有名なのが，シャネルのスーツとバッグ（シャネルの5番って香水も有名だけど）。なので，三越のシャネル ショップに到着した私はとりあえずバッグコーナーへ直行。そこで見たのはオレンジや黄緑といった素敵な色遣いのバッグです。財布，ハンカチ，口紅，携帯電話を入れればいっぱいになりそうな小さいバッグたちには20万円とか24万円とかいった値札がついている。なに？ シャネルのバッグは高いとは聞いてたけど，20万円がデフォルト値～！ パソコンが買えちゃうじゃん，とビックリし，バッグがダメならスーツだと洋服コーナーへ移動。明るいパステルカラーの生地に金の縁取りが上品でゴージャスなスーツが素敵だったので，値札を見ると怒涛の60万円。黒のシンプルなジャンパースカートは半額の30万円。ブラウスはけっこう安くて15万円だぁ。ふ，ふざけるな～シャネル。

しかし，私は負けたくないっ。なので，今度はアクセサリの類にターゲットを変えました。シャネルのアクセサリはメッキやイミテーションの宝石をあしらった，大ぶりのアクセサリが主流です。イミテーションならそんなに高くはないだろうと思い，金色の留め金の先に赤い七宝が輝いているイヤリングに目をつけ，値札が見えなかったので（アクセサリ類はガラスケースに陳列されていた

のです）店員さんに価格を聞いてみました。

「こちらは最新作で7万円でございます」

「はあぁ，7万円ですか。目をつけていたMacのグラフィックアクセラレータが買えちゃ……，いや，高いですね～」

と，完璧に打ちのめされた私が名残惜しそうにイヤリングを眺めていたところ，店員さんが親切にささやいてくれたのです。

「あの，2万円台のイヤリングもありますが，ご覧になりますか?」

「!? 見せてください」

かくして，いぶし金にシャネルのロゴが入ったシンプルな丸いイヤリングを2万円で購入して，とりあえずはシャネルを買ったってことに満足した私は，シャネル集めなんてのは忘れることにして，パソコンとゲームに明け暮れる普通の人生を送ろうと反省したのでした。

海外旅行でシャネルが復活

一度シャネルに挫折したときから1年くらいあとのこと，私はグアムに遊びにいきました。グアムには泳ぎにいったのですが，ついでに，いい加減古くなったバッグと財布を買い換えようと思いフェンディ，ヴィトン，ディオール，エトロ，ベルサーチ，コーチといったいろいろなブランド店を見て回りました。でも，なぜかこれといった商品にめぐり会えません。バッグも財布も買えないでトボトボとグアム市内を歩きまくっていた私は，行くのを避けていたシャネルをのぞくことにしました。

でもそのときは，ちょっと見るだけでバッグも財布も買わないようにしよう，って気持ちだったのです。ところが，シャネルのお店に入った途端に綺麗なオレンジ色のスカーフが目に入りました。当時は円高時代だったので，価格もそれほど恐ろしいものではありません。それを見た瞬間に店員さんを呼んで，

「あのスカーフください」

と一気にスカーフを購入したのです。店員さんは気さくだけど上品な日本人で，愛想よく対応してくれました。

「ほかになにかいりませんか?　バッグもお品を取りそろえておりますよ」

ふと，バッグコーナーを見ると，色がワインレッドでB5の本が楽勝で入るバッグがあるではないですか。バッグを持ったときの私のシルエットも綺麗に見える。これだ，これこそ探し求めていたバッグだ。バッグ自体もそのバッグと私のコンビネーションも綺麗に見える。考えてみると，気にくわなかった他ブランドのバッグはバッグ自体はいいものなのですが，私が持つとちぐはぐ感がぬぐえなかったのです。やはり，私とシャネルはベストパートナーになる運命だったんだ。

「あ，じゃあバッグもひとつ」

「ほかにお財布などはどうでしょう?」

店員さんがおすすめの財布は黒の丈夫そうな革にシャネルのロゴが大きく入っているシンプルなもの。シンプルかつゴージャス。これこそ私が探し求めていた財布だ。

「あ，じゃあ財布もひとつ」

「ほかにアクセサリもきれいな新作がございますよ?」

アクセサリコーナーではシャネル特有の大きなイミテーションストーンをあしらったキラキラ光るイヤリングを見せてもらった。以前購入したいぶし金イヤリングとは対照的に派手なもの。しかしつけてみると無色のストーンにいやみがなく，遠くからでも十分わかるキラキラ感が男をすーっと引き寄せそう。これこそ，私が探し求めていたイヤリングだ。

「あ，じゃあイヤリングもひとつ」

「ほかにスーツとかは……」

「あ，スーツはまだ心の準備が……」

急に我に帰って一気に選んだ品物分を精算。買った分を合計すると円高を考えるとかなりの額だったけど，心はグアムの空のように晴れ渡っていました。だって，ようやく自分のペースで高級ブランドを攻略できたんですからね～。日本橋三越のときは価格にとらわれて，自分に似合うものを選ぶという重要なことを忘れていたようです。そのため，

「ん～，シャネルっていってもなにかイマイチのような～」

と思いこんでしまっていました。ところが，あらためて自分に似合うものを身につけると，

「こんなにゴージャスで，自分を素敵に演出してくれるブランドなんてほかにはない」

って気持ちに変わっちゃいました。アクセサリひとつで，シンプルなスーツも華やかなものに変身するからです。

もう，こうなると恐ろしいと思っていた価格すら気にならなくなります。こうしてシャネルの魅力にとりつかれた私は『シャネルを買うときには，価格を見ないで一気に購入しましょう』という，シャネル購入の法則ってのを作りました。でもね，これを率先して実行しちゃうと，お金がなくなっちゃうし，ほかの趣味にも影響が出ちゃうから『ボーナスや原稿料って臨時収入が入ったときに』と条件付けはしたのですけどね。

シャネルショウへの招待券

日本に戻りしばらくたった頃，私は雑誌で見つけた紫色のシャネルのイヤリングがほしくなりひさびさに日本橋三越シャネルへ行きました。ところが，ほしいイヤリングが見つかりません。店員さんに聞いてみると，

「人気商品は入荷するとすぐに売れちゃって品切れになるんですよ～」

との答え。なぜ，こんなに高い商品がすぐ売れるの～？　店員さんの話では，シャネルの商品は正

規の代理店でも数多く入荷しないそうです。そのため，シャネラーならほしがりそうな人気商品は，すぐ売れてしまうそうです。

むむ～，こんなところで「ほしいものは即行で買わないと次はなくなっている」というアキバの法則が生きているとは……。変なところでシャネルとアキバの共通点を発見。アキバと違う点は，シャネルは取り寄せができないところです。

とにかく，負けてたまるかと，まんまと品薄商戦に乗せられた私は，ある日，夏用の涼しげなイヤリングを衝動買いました。会計を待っているときになにげなく洋服のカタログを見ていたら，会計から戻ってきた店員さんがそれを見て，

「洋服とかもご興味ありますか」

と話しかけてきました。

「ええ，ほしいですね～(金があればね)」

「なら今度，帝国ホテルでプレタポルテ[*3]のファッションショウがありますので，ご出席なさいませんか？」

「ええっ，いいんですか?」

「それはもう，よかったらお友達の分も一緒に席をご用意しますよ」

「あ，じゃあ2人分ってことで」

かくして，ひょんなことから憧れのシャネルのファッションショウへのご招待券をゲットしちゃったのです。あ，でも平日開催。会社は半休だわ。

シャネルのファッションショウ

シャネルのファッションショウは帝国ホテルの「富士の間」で開催されました。開始15分前くらいに帝国ホテルに到着し，会場に入ってビックリ。黒服のおねーさん，おばさま，おじさま(おにーさまはいなかった)がズラーっと並んでお客を迎えているではないですか。それも全員黒服（シャネルの制服なんだけどね)。一歩間違えれば○くざの総会だぁ。おまけに，その横のコーナーにはシャネルの服がズラーっと並んでいる。

「うわぁ，こりゃあとんでもないところにきてしまったような気がする……」

と，つれにそっとつぶやき，

「私もちょっと緊張しております」

と，つれも答える。

それにしてもシャネルの店員って若い女性ばっかりだったのに……？　あらためて招待券を見直すと，このファッションショウは日本橋三越が主催。つまり，マスコミ向けの発表会ではなく，三越が企画した即売会が目的のショウだったのです。現地に行くまで気づかない私もそうとうアレなんですが，なにせショウに招待されたのも初めてだったものでして。

「なあに，買わなくても東京湾に沈められることにはならないでしょ」

と開き直り遠慮なくズカズカと会場に入りました。会場内を見回すとお客の数はざっと100名ほど。そのほとんどが女性でありシャネルで装ったシャネラー。2～3人男性もいますが，どうやらパトロンのご様子。店員さんからはショウが見やすい席を勧めてもらって，帝国ホテルのボウイさんからコーヒーをサービスしてもらい，ショウが始まりました。

ショウの目的は即売会なので，モデルさんは服の番号札を持ちながら歩いてきます。お客はモデルが着ている服と番号札をチェックして，あらかじめもらってあるカタログで型番と価格をチェック。気に入った服があったら，ショウ終了後，販売コーナーで試着してその場で買えるシステムです。しかし，番号札を無視すりゃあ立派なファッションショウ。それもモデルの足音が聞こえるくらいの間近な特等席でです。

「あの服の生地すんごくきれい～」

「うっ，あのモデルの足にタトゥーが入っている」

「ああ，アクセ[*4]がゴージャス。あそこまで大きいと絶対10万円はするぞぉ」

「モデルさん，胸ないね～」

と，特等席で十分シャネルの夏の新作(これは今年春の頃のお話)を堪能したのでした。ああ，正規の代理店で買物していてよかった。さすが天下の三越だ。

ショウが終わると，ほかのお客さんはササっと姿を消しました。感動の余韻が残っている私たちはコーヒーを飲みながらのんびりしていたのですが，つれがシャネルのロゴ入り鉛筆を指しながらいいました。カタログのチェック用にあらかじめテーブルに置かれてあったものです。

「この鉛筆もらってもいいんでしょうかね～」

「ええっ，鉛筆持っていっちゃうなんて，なんか貧乏くさくない?」

「でも～，ほかの方も～，持って帰っているし～」

テーブルの上を見ると，価格表と鉛筆が消えている。さすが，シャネラー。シャネルのロゴが入っていれば，鉛筆だろうとなんだろうとほしいもんなんだ。

ショウが終わったらさっさと帰る予定の我々でしたが，せっかくきたのだからと即売コーナーものぞくことにしました。即売コーナーに行ってみると，そこはすでに気に入った服を我先に買おうとしているハイソなお客様の戦場と化していました。どえぇ，いいものは即行でゲット……の法則がここでも生きている。どおりで，ショウが終わったらすぐにいなくなっちゃったワケだ。価格からして30万円から70万円の服が飛ぶように売れ，妙に着飾ったおばさまが，これまた，らしいおじさまたちに挨拶しまくっている。いやー，こういう世界もあるんですね～。いつかお仲間に入りたいものです……。

とのことで，シャネルにまつわるお話でした。

なにごとも勉強です。とにかく体当たりでお店でものぞいてみてはどうでしょうか？　写真で勉強するのもいいですが，本物を見ると理解度がさらに増しますよ。本当は買うのがいちばんですが，見るだけでも勉強になります。ひとりじゃお店に入りにくいって方は，彼女か学校/会社の女性でも誘ったらどうでしょうか？　おねだりされちゃってもしかたありませんがね。では。

＊1　パリ コレクションの略で，パリで開催されるさまざまなブランドの発表会の総称。でも幕張メッセみたいなところで一気にやるんじゃなくって，あちこちの場所でブランド別に開催される。もちろん開催地がミラノだとミラノコレクションというのだがミラノコレとはいわない。なんか不思議。

＊2　発注者に合わせて作るオーダーメイドの服。

＊3　既製服。いわゆる吊しの服のこと。

＊4　アクセサリーのこと。「シャネルのアクセ買っちゃったの～」と使う。

参考資料

FASHION NEWS（ムック：流行通信社）

CHANEL BOUTIQUE（非売品：CHANEL PARIS）

ココ・シャネルの秘密（文庫本：早川書房）

Oh!X LIVE in '98

X68000 Z-MUSIC ver.3.0用MU100対応
MOON OVER THE CASTLE
須賀 真

X68000 Z-MUSIC ver.2.0用SC-88Pro対応
道(オリジナル曲)
河野匡格

MOON OVER THE CASTLE / ANDY'S ~ GRAN TOURISMO Version ~

復活Oh!X LIVE第1号の曲は，去年末に発売されて以来，大人気のPlayStation用ドライブゲーム「グランツーリスモ(GT)」でもメインテーマに使われた「MOON OVER THE CASTLE」です。

この曲は，平成のテーマソングライター「T-SQUARE/安藤まさひろ(MASAHIROANDOH)」が作編曲を行った作品です。T-SQUAREといえばF1のテーマでお馴染みの曲「TRUTH」が有名ですが，とにかくこのバンドはキャッチーなメロディを聞かせてくれます。曲の主題となるメロディ……いわゆるフックラインはどの曲もみんな似通ってはいるんですが，そうとわかっていても一度聞くと頭から抜けなくて，ふと我に返ると口ずさんでいる自分に気づいたりすることがあります。こういう音楽は各界に相当都合がいいようで，T-SQUAREの曲はスポーツ競技とか，テレビ/ラジオ番組のテーマとかにやたら採用される傾向にあります。

こうした「一度聞くとつい口ずさんでしまう」という音楽は昔ながらのゲームミュージックにもいえる特徴です。そういう意味で「T-SQUAREは一度ゲームミュージックをやったら面白いんじゃないかな」と思っていた方も多かったでしょう。そしてこの夢はPlayStation用RPG「アーク・ザ・ラッド」で実現されました(アルバムCDも出ています:"Arc the Lad"AntinosRecordsARCJ12)。今回のGTのサウンドはT-SQUAREのゲームミュージック界進出タイトルとしては2作品目にあたることになります。

須賀さんが制作したデータはZ-MUSIC ver.3.0専用の演奏データで，対応音源はYAMAHA MU100/128シリーズです。実際の演奏はCD-ROMにも収められています。聞いてもらえれば一発でわかると思いますが，とてつもない完成度に仕上がっています。T-SQUAREのMIDIデータ集というのは数多く発売されていますが，この曲に限っていえば「そのどれよりも完成度が高い」といえるかもしれません。特に，ギターパートに注目してください。フレット上を高速に行き来する指が見えてきそうな演奏です。もちろん原曲と聞き比べれば，どちらが人間の演奏かわかるかもしれませんが，ぱっと聞いただけではDTMの演奏とは思えません。この秘密はなんでしょうか。

Z-MUSICユーザーの方はぜひともリストを見ていただきたいのですが，この「人間くささ」の秘密はMU100が持つ「インサーションエフェクト(INSERTION EFFECT)」機能，そして須賀さんのシーケンステクニックにあるといっていいでしょう。

インサーションエフェクトとはMU100/128の特徴的な機能のひとつで，デジタルエフェクタを特定のMIDIチャンネルに専属でかけることができます。ギタリストは自分独自のギターサウンドをエフェクタで作り込みますが，須賀さんは，インサーションエフェクトの「ディストーション」「オーバードライブ」を駆使し，そのパラメータを極限まで調整してT-SQUAREのギタリスト安藤まさひろのギターサウンドを再現しているわけです。

トラック13がリードギターのパートです。ところどころにNRPN(ノンレジスタードパラメータナンバー：MIDI規格で未定義のパラメータで音色定義などでよく使われる)の指定がしてあり，演奏表現に応じたエフェクト効果を1つひとつ丹念に作り込んでいるのがわかります。あまり使用例が多いとはいえないZ-MUSICの特殊機能「アフタータッチシーケンス」，そして「PUSH」「PULL」の2通りのポルタメント演奏機能の使い分けを行っているのも，Z-MUSICユーザーには参考になるでしょう。

X680x0と歌ってみました

2曲目は河野匡格さん作曲，池田尚隆さん作詞のオリジナル曲「道」です。タイプとしてはスローなバラードで，レクイエムともいえる落ちついた静かなイメージが漂った曲です。

曲データ自体はZ-MUSIC ver.2.0のデータで，演奏にはX680x0とZ-MUSICが必要です。対応音源はローランドSC-88Proですが，SC-88/VLでもそこそこ聞くことができます。この曲も実際の演奏が付録CD-ROMに収録されていますので聞いてみてください。

聞きましたか？ 冒頭で「作詞」というキーワードで薄々感づいていたかもしれませんが，そうです，この曲には歌が入っているんです。CD-ROMにはZ-MUSICが制御するSC-88Proのバックバンド演奏にあわせて女性のボーカリストが歌ったものを収録しています。今回の作品では演奏データであるZMSファイルと，この最終的な演奏形態であるオーディオファイルも投稿してもらっています。CD-ROM収録形態が可能になった新Oh!Xならではの採用例といったところです。

さて，曲のほうですが，ボーカルもの……ということで，メロディ主体の曲になっています。最近のハヤリなんでしょうか，スローな割には展開の予想がつかみにくい非常に技巧的な構成になっているようです。なんとAメロのあとに突然転調がきます。Bメロからエンディングに向かい1コーラスが終わり，続いて2コーラス目でAメロに返りますが，このとき元の調に戻ることになります。聞いているほうとしてはここでも「また転調？」という感じを受けることでしょう。初めて聞くと「転調に次ぐ転調？」というイメージを抱くかもしれません。が，よ～く聞くと主題はAとBの2つだけということに気づくはずです。メロディだけを追うとそれほど不自然ではありませんが，普通のバラード曲として聞くとちょっとびっくりさせられるかもしれません。

曲の進行はやや独創的ですが，各パートの作りは非常にオーソドックスです。ボーカルパートを補うハーモニーを作るストリングス隊や，ボーカルパートとユニゾンコーラスを決めるピアノやギターパートの作り込みは，「歌もの」の曲を作っている人にはとても参考になると思います。

曲データのZMS自体もシンプルですが，やるべきことは全部やっている優等生的なデータです。ピアノやギターの演奏情緒は肌理細かくZ-MUSICの機能のひとつであるベロシティシーケンスを使っていますし，ギターのソフトな指使いは同じくZ-MUSICの和音機能のバリエーションであるアルペジオ機能を駆使して表現しています。

そうそう，忘れるところでした。このデータはZ-MUSIC ver.3.0でもver.2.0でも演奏できますが，ver.3.0で演奏するときは必ずリスト最後の，(s)コマンドを削除してください。なぜかこの曲データには演奏開始指定の(p)の前に(s)という演奏中止の命令が書かれているのです。この変更をしないとver.3.0では演奏が開始されません。

(西川善司)

Step to the Black Arts

最近はプログラミングは流行らないらしい。
西暦2000年になるとプログラマがどれだけ足りなくなるのかは忘れたが，いまや危機意識を持っている人などいない。
アプリケーションやツールなど十分すぎるほど供給されているし，ゲームなどは専用機で山ほどできる。
こういう世の中ではプログラムなどを作っていると，なぜか肩身の狭い思いをする。
「そんなもの一般ユーザーがやらなくていい世の中になったのに，こいつはなにをやってるんだ？」と奇特な人を見る目で見られてしまう。
やらなくてもいいのは確かだが，「やってもいい」というのも確かなのだ。
そして我々は好きでやっているだけの話だ。プログラミングは知的なホビーだ。
仕事でするプログラミングは苦痛なものもあるかもしれないが，楽しむためのプログラミングだってある。
そしてパソコン自体をもっと楽しむためには不可欠なものなのだ。
現在，あなたはパソコンに触っていて楽しいだろうか？　現在のパソコンのパワーは強力だ。
その力を駆使すれば，パソコンは現在さまざまなアプリケーションが見せている世界とは
もっともっと違った姿を見せてくるだろう。それはきっとできるはずなのだ。新しく本誌を始めていくに際して，必ずしも準備は十分ではない。
もっとノウハウの蓄積も必要だ。しかし，それが整うまで待ってはいられない。時代はさらに先に進んでしまうのだ。
パソコンを真の意味で自在に操れるようになる日まで，まずはできるところから進めていきたい。
そして，このようなホビープログラミングのための場所を確保していきたい。ではその一歩から始めていこう。

プログラミングセクションは３つのパートから構成されている。
Level1　プログラミング入門編
Level2　応用プログラミング入門編
Level3　実践的プログラミング編
それぞれ自分の興味のありそうなところを見ていってほしい。

VisualBasic CCEから始めてみよう

中野修一/Nakano Shuichi

Windowsを使っている人は多い。しかしそこでプログラムを作っている人は極端に少ない。

まあ，自分で作らなくたってなんとでもなる。インターネットにつないで探せば，誰かがいろんなもの作ってるので，自分で作る必要はない。それはそのとおりかもしれない。

急ピッチで進むOh!X編集作業を見ていると，機種非限定ということでいろんなデータが飛び込んでくる。なかでも，データのやり取りで，いわゆるMacバイナリというのが邪魔になる。これはMacファイルの先頭についた128バイト長のヘッダだ。ファイル本体の前についているので，JPEGファイルなどでもそのままでは読めない（ローダ側で対応させろよなという気もしなくはない)。なんか無茶苦茶素人っぽい例で申しわけないが，その話をしてみよう。

ちなみに，似たようなファイルのやり取りはこれまでにもあったわけだけど，それまでどうしてたかというと，私の場合，X68000でツールを作って処理して，またWindowsに持ってきていた。我ながらバカらしいことをしていたと思う。

世の中この手のものを探せばないわけはない。探してみる。あった。日本最大のダウンロードサイトであるvector.co.jpで探してひとつだけ。落としてみてさらに驚く。Win32を使っていて，なぜDOSアプリ……。

きっともっといいものもあるだろう。ないはずがない。あるいは，NIFTYのライブラリのほうがマシなのかもしれないが，はっきりいって，探すほうが面倒だ。Windowsの世界は広大で，探せばたいていの用途でいいツールもあるのだろうが，うまく探さないとイマイチなツールが山ほど見つかることになる。

加えて，そういったツールを使っていて思うことがある。いっちゃいけないことかもしれないが，あえていう。

「なんか違うぞ」

手に馴染まないのだ。「そこでいちいちダイアログ開くか?」，「そんなことをいちいち確認するんじゃない」

Windowsだからかねぇ～とも思いつつ，どっちかといえば，たぶん私の感覚のほうがおかしいのだろうというのは自覚している。

前置きが長くなったが，結論は簡潔にしよう。いいのがなければ作りゃあいい。

■VisualBasicの導入

プログラムを作るにはそれなりの環境なりツールなり言語なりが必要だ。

ここではWindows上で我々にもっとも身近な開発環境であるVisualBasic（以下VB）を使ったプログラミングについて解説していきたい。言語はなんでもいいんだが，マイクロソフトといえばBASICが基本だ。WordやVBではVBAというのが走ってるし，IEでもVBScriptが走る。ビル・ゲイツといえば，タイニーBASICを移植して財を成した人だ。Windowsの成功もVBによるところが少なくない。Windows98だってOffice97だってBASICで書かれている（嘘)。

それはともかく，まず，

「VBでゲームを作ろう」

と思う人がいるだろうが，ここではその話はあまり扱わない。もちろんそれはできないわけではないし，別のテーマとしては面白い。しかし，ここ

VB5CCEとは？

COLUMN

さて，VBといってもいくつかエディションがあるのだが，ここではVB5CCEを使用する。これはVB5.0をベースにしており，ActiveXコントロールを作るためにマイクロソフトが無料で配布しているものだ。もちろん機能制限はある。単独の実行ファイルも作れない。それでも無料で配布されているプログラミング環境というのは魅力的だ。Web上では配信されているのだが，VBのバージョンが変わるということで，CD-ROMには収録できなかった。もちろん，VB6CCEが出るというちゃんとした話があるわけではない。おいビル～っ，なにやってんだよぉ。

ちゃんとしたパッケージを買うのなら，

スタンダードエディション

というのがあって，とりあえずはこれでいいはずだ。ほかには，

プロフェッショナルエディション

というのもある。むしろ，これが標準的なパッケージだと考えていい。普通はこれで十分だ。むしろほとんどいらないようなものがたくさん入っていると感じる人もいるかもしれない。

エンタープライズエディション

とかいうのは個人には関係ないものだと思ってもいいだろう。業務用のアプリなどを開発するのには便利なものが入ってるらしいのだが，そういうのをVBで作る人間にだけはなっちゃいけない気がする。悪いことはいわないからVC++を使いなさい（個人的見解)。

VB5CCEはVB5のサブセットだ。プログラムの話をするのなら別にVB5自体でもよいのだが，VB5CCEでもある程度のことはできるし，なにより私はVB5を持っていないし，本当に必要になるまで買う気もない。私は貧乏なのだ（VC++ ProfessionalEditionは買っているが，それはまた別の話だ)。

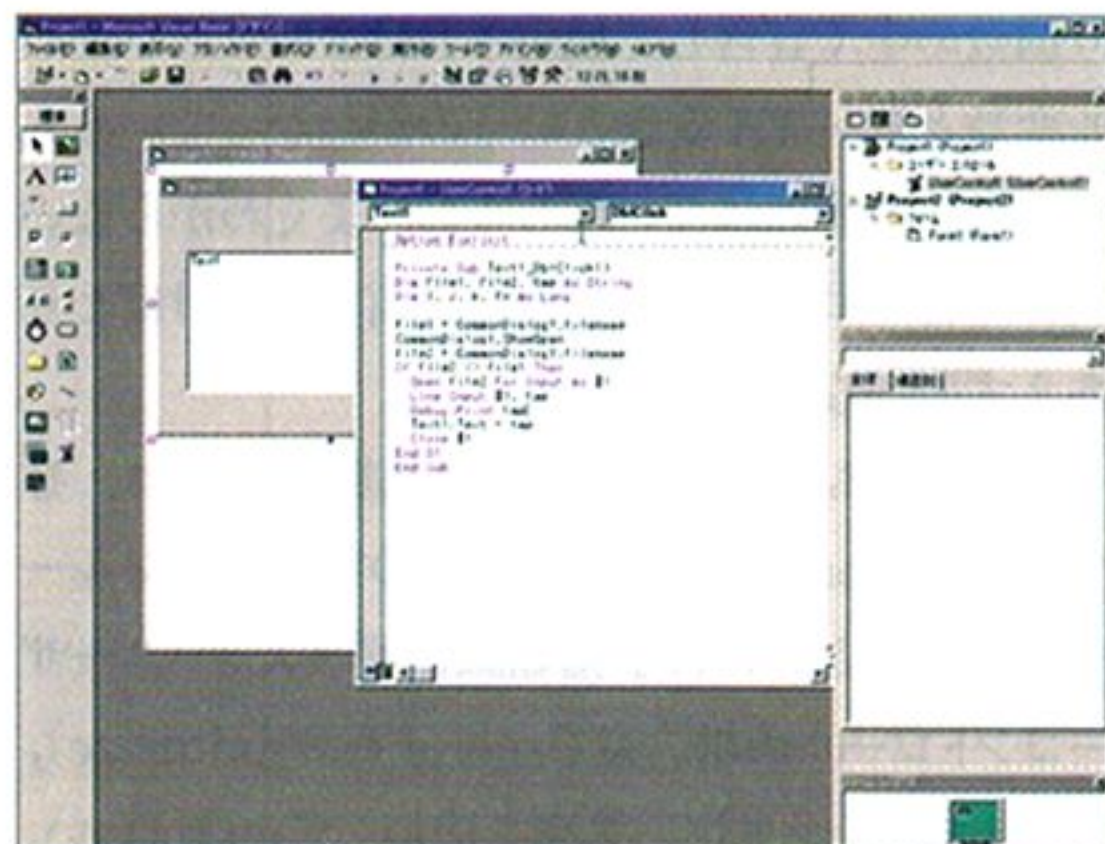
さて，VB5CCEで想定されているActiveXコントロールというのはOCXがWindowsDNAな感じでWebページに張りつけたりすれば動くというものだが，それがなんなのかを深く考える必要はない。張っておけば動くのでアプリと変わらない。そういうものだ。ショートカットがHTMLファイルになっているだけだと思えばいい。IE4.0を使えばウィンドウの背景やツールバー内で動作させることもできるし，IE5.0ではウィンドウレスで起動するものも作れるようになるはずである。インターネット環境でActiveXを使うのはあまり感心しないのだが（個人的には)，ローカル環境で使うのならなんも問題はないだろう。

となるとActiveXというものを説明しなければならないのだが，これは簡単にWindows全体で共通に使える部品だと思っておけばいいだろう。DLL以来のダイナミックリンク（そのライブラリが必要になった時点でメモリに読み込む）というのが，インターネットを使ったものにまで拡張されてきたものだ。

ハードディスク上になくても指定されたサイトのディスク上にあればさまざまな機能がインストールされて使用できるという具合になってきている。

まあ，これはあくまでインターネットに接続しているというのが前提だ。専用線でつなぎっぱなしにしている人とダイアルアップでテレホ時間にひしめきあっている人では全然環境は違うので。どれくらい現実世界から遊離した思想かはともかく，基本的な考え方は間違ってはいないだろう。将来的にはみんな専用線が当たり前になるのだろうし（5年は無理でも10年後くらいには大丈夫かなあ……。5年も経てば全然違うものが出てきてるかもしれないけど)。

すべての部品を独立したオブジェクトにして，さまざまなパーツの組み合わせでいろんなものを作れるようにして生産効率を上げよう，というのがたぶん基本になっている。Webページに張りつけるコントロールも，ほかのもので使われるオブジェクトも基本的には差がない。さらにデータのやり取りなどでOLE2とかいろいろあるのだが，いろいろ調べていけばWindows自体の構造が見えてくるはずだ。VBを知ることはWindowsを知ることになるというのはこういう意味もある。

図1　VisualBasicの基本画面

ではプログラミングの基礎の基礎，開発環境に慣れ親しむところからやっていくので，ゲームについては別の機会に話したい。VBを日々ツールを作ったり，日常的なプログラミングを楽しむための環境だと考えてほしいわけだ。そういうのが手に馴染んできたらゲームくらいなんとでもなるだろう。

とにかく，VBのよさは「VB5CCEというのが無料で配布されている」ということに尽きる。もちろん，この分野ではヒット商品なので美点はたくさんある。ちょっとした業務用ソフトならちょいちょいするだけで作れてしまう。Windows用アプリの開発が簡単ではないことを思えば，絶賛されてしかるべきソフトだ。ただ，業務用ってのが問題で，ゲームとかで必要なイメージや音声の扱いになるとからきしダメだった。まあ，これはVBの責任ではない。そもそもWindowsがそういうものだったのだから。

とにかく，現状では無料で入手できるWindows用プログラミング環境として絶対的な存在となっている。これを使わない手はない。

さて，「CCE」というバージョンであることの注意もあるのだが，ここでは「コントロールを作っていこうね」という模範的なアプローチでいくのであまり問題はないだろう。スタイルとしては正しいと思うし，ならばCCEがあればこと足りる。うん，いいよビル，これは。

■ツールを作る

なにはともあれBASICだから，結構多くの人はなんとなくプログラムを作るくらいのことはできるのではないかと思う。それとも8ビット機の頃からやってた人だけだろうか……とちょっと不安はあるが，最初なのでプログラムを作るというのはどういうことかから始めてみよう。

入力
↓
演算
↓
出力

というのが一般的なプログラムの流れのひとつだ。なにかのデータを渡して，それをプログラムで加工し，なにかに出力する。そんな流れをなんとなくイメージしてみてほしい。ただ，ウィンドウプログラムの場合これがちょっと変わってくる。

入力部分がキーボードやファイルからの入力だけではなくて，OSによって管理されているさまざまな「イベント」が引き金になることが多い。むしろなんらかの入力はイベントというかたちでプログラムに渡されると考えたほうがいいだろうか。

イベントには「キーを押された」「マウスカーソルが乗った」「クリックされた」「よそから値を変えられた」ほかたくさんの種類があり，それぞれに応じた処理ルーチンが呼び出される。Visual Basicを使う際には伝統的なBACICプログラム様式でも大丈夫なのだが，こちらの流れを理解しないといけない。決して難しいものではないので，とにかく慣れることから始めよう。

■Windowsでのプログラムの流れ

Windowsでは，さまざまなものがさまざまなイベントを発生する。それに対応する処理ルーチンを用意していく感じでプログラムはできあがる。VBでもそれは同じだ。

こんなのは解説を読むよりやってみたほうがわかりやすい。とにかく，VB5CCEをダウンロードしてインストール実行してみてほしい。ちょっと前の雑誌とか，VBの解説書なんかについていることもあるので，そっちを探すのが楽かもしれない。

実例だ。

VBを起動すると最初にどんなタイプのプロジェクトを作るのかと聞かれてくる。VBがプログラムを管理するときの単位なので「プロジェクトってなに？」というのは気にしなくてもいい。どれでもいいといえばどれでもいいので，EXEファイルを指定する。「とりあえず」のときはEXEにしておけば間違いはない（「とりあえず」ではないときのことは後述する）。

ではフォーム（最初に出てくる台地）に左端のバーから縦スクロールバーのアイコンを選んで2つ置いてみる。

VBのウィンドウの左端に並んでいる各種ボタンやスクロールバーなどの部品をまとめて「コントロール」と呼ぶ。コンポーネントと呼ばれることもある。とにかく，一定の機能を持った部品の集まりだ。これらの部品を使ってプログラムを組み立てるのがVBの基本だ。

で，とりあえず部品を配置した。このまま実行（ボタンを押す）してもいいのだが，このままではなにも起きない。スクロールバー1をダブルクリックしてみてもらいたい。ウィンドウが開いて，

```
Private Sub VScroll1_Change()
End Sub
```

というのが表示されたことと思う。名前を見てわかるとおり，これはスクロールバーの値が変化した場合に呼び出されるルーチンとなる。ここに処理を書いてみよう。

```
Private Sub VScroll1_Change()
Debug.Print VScroll1.Value
End Sub
```

としてみる。ちなみに，大文字，小文字は判別されないので，すべて小文字で打ち込んでいけばよい。登録された文字列だったら，勝手にキャピタライズされるのでミスタイプ発見にも役立つ。さらにいえば，

```
debug.p
```

と打ち込んだところで候補が出てくるはずなので，実際のキー操作は，

```
d e b u g . p Tab
```

を順に打ち込んでいけばいいことになる。

ここで使ったDebug.Printは多用される命令で，デバッグ用のイミディエイトウィンドウにデータをコンソール出力するためのものだ。オールドタイプなBASICのPRINT文と同じ感覚で使用できる。

さて，この状態で実行すると，スクロールバー1を操作するごとに数値が表示されるのがわかるだろう。スクロールバーが0～32767までの値を返す物体であることがわかる。

ここで出た0と32767という比較的お馴染みな数値は，実は右端にあるプロパティウィンドウに見えるVscroll1のMinとMaxの値そのものである。スクロールバーはここで指定された範囲の数値を位置に応じて返す機能を持っている。設定範囲は－32768～32767という実にx86な感じの16ビット仕様だ。

ということで，プログラムをちょっと変更してみる。

```
Private Sub VScroll1_Change()
Debug.Print VScroll1.Value
VScroll2.Value = 32767 - VScroll1.Value
End Sub
```

これで，スクロールバー1のほうを動かすともう一方が逆の動きをすることがわかる。これは「VScroll2のプロパティをVScroll1のイベント処理ルーチンで変更した」ということになる。

このほかにも，このプロパティウィンドウ内の値をいじることでオブジェクトの挙動を制御することができる。この「プロパティ」というのが，機能制御のためのパラメータ群であることがなんとなくイメージできただろうか。

まとめてみよう。コントロールは必要に応じてイベントを発生するので，それに応じた処理ルーチンを用意するといろんな処理ができる。また，

オブジェクトのプロパティを書き換えることでその動作を制御できる。

使い方を見ると，「VScroll1」というオブジェクトで「Change」というイベントが起きると，自動的に「VScroll1_Change ( )」というサブルーチンが呼び出されるようになっている。「VScroll1」の内部変数である「Value」にアクセスするときは，「VScroll1.Value」と記述する。これ丸ごとで変数として機能しているのがわかるだろう。この辺はいちばん基本になるところなのでしっかり押さえておこう。

とあるコントロールがどんなイベントを出すのかは，コードウィンドウの左上でオブジェクトを指定したうえで，右上のドロップリストを眺めれば一覧できる(正確にはちょっと違うのだが)。また，どんなプロパティが用意されているのかを知るには素直にプロパティウィンドウを見ればよいだろう。もっと詳しく知りたいときは，メニューの[表示]から[オブジェクトブラウザ]をいうのを開き，「クラス」というところで知りたいアイテムを選択し(この場合はVscroll)，右のメンバ一覧を見るとよい。種類はアイコンで区別されているのだが，

がイベント

がプロパティ

がメソッド関数

となる。それぞれ選択すると，下に解説が出てくるので，どんな動作をするのかがわからないときにはこれが便利だ。なお，メソッドについてはまだ解説していないが，プロパティのような値を介さなくても実行できる命令のようなものだ。プログラムを動かしていくうえでこれらの3つの概念を頭の片隅に入れておいてほしい。あとは実践を通して覚えていけばいい。

■プログラムの基本

イベントが起きて処理が移るのは，所定のサブルーチンとなる。あ，ルーチンってのは仕事って意味で，処理の1単位みたいなもの。下請け仕事用のプログラムがサブルーチンということになる。プログラムには普通メインルーチンがひとつあって，その中から必要に応じていろいろなサブルーチンが呼び出される……という感じに作られることが多い。VBで普通になにか作ればイヤでもそうなる。

単純な命令を組み合わせて複雑な処理を効率よくこなすための方法論はいくつも存在する。ここでの本題ではないが，基本的に，

「構造化する」

とか，

「モジュールにする」

というのは一時とても流行した考え方だ。プログラミングの基本といってもいい。難しく考える必要はないが，大きなものを作るときにこんがらがらないようにきちんとやろうってことだ。一度くらいはその手の解説書を眺めてみたほうがいいかもしれない。こういうのが，なんとなく頭の片隅にあれば，結構プログラムというものに対する考え方がわかってくるものだから。

さて，VBの基本に戻ろう。処理の中身はどうやって書くのかというと，VBに実装されている命令を使う。BASICが理解できる人ならたいていそのまんまいけるし，わからない人はヘルプをよく読むか，文法リファレンスの載っている解説書は最低限必要になる。ここでは命令の種類などについては触れないので各自で勉強してほしい。

次にVBのプログラムではさっき述べたような，個々のイベントなりに対応した処理しか書けないのか？　というとそんなことはない。イベント処理部分は自動生成されたサブルーチンの形態になっていたことはおわかりだろうか？　それと同じように自分で名前をつけたサブルーチンを作れば，それは定義済みの命令語のように使用できる。

プログラム全体で使用する変数はプログラムの先頭部分で宣言しておく。あとはサブルーチンなり関数の中で宣言して使用すればよい。グローバル変数とローカル変数というのがわかっている人ならなにも問題はないだろうし，よくわからない人は全部プログラムの先頭で宣言しておけばとりあえず問題ないだろう。

プログラムが起動したときの初期化処理は，必ずForm_Loadルーチンとなるので，メインルーチンを組むときには予備知識が必要だが(ユーザーコントロールの場合はUsercontrol_Initialize)，あとはルーチン単位に処理を追加していけばいい。

■処理の実行

VBでは具体的になにか処理を行う際にいくつかの方法がある。

・BASICの組み込みステートメントや組み込み関数を使う

これは当たり前の話だ。

```
Beep
```

と書けば音が鳴る。従来のBASICを使ったことがあればなんの疑問もないだろう。

・オブジェクトのプロパティを変える

そのウィンドウを構成している各部品の属性を変更して処理を進めるというものだ。

```
Label1.ForeColor=Vbred
```

関数とサブルーチン

COLUMN

BASICでは使える命令が決まっており，ステートメント(命令)と関数の2種類が存在する。

```
Beep
```

はステートメントだし，

```
Sin ()
```

は関数だ。括弧付きのものが関数……ってのは目安だけど，

```
a$=mid("abc",1,2)
```

は関数でも，

```
mid(a$,1,2)="abc"
```

はステートメントと見なされる。

BASICでは値を返すものが関数で，なにかの処理を実行するものがステートメント……というのが基本だった。しかし，BASIC以外の言語では関数は値を返すだけでなく，「副作用」を伴っていろいろな処理を行うものという認識も出てきた。さらにいえば，関数はなにかの処理を行って，実行結果を返すものという認識だ。

ユーザープログラムでは，

```
Sub 名前()
    処理
End Sub
```

となっているものがサブルーチンで，

```
Function 名前()
    処理
End Function
```

となっているものが関数だ。伝統的にはBASICではサブルーチンを使用することが多い。逆に関数しかない言語というのも存在する。どちらかというと，関数を使ったほうが綺麗な気はする(個人的印象)。数値演算の実行結果とかだけではなく，たとえば，ファイル書き込みの結果など，メインプログラム側が成功したのか失敗したのかレポートを必要とする場合がよくある。こういうのを効率よく記述しようとすると関数が多用されてくることになる。ほかの言語に馴染んでいる人ほどこういった関数の使い方をする。

なにが関数でなにがサブルーチンかという区別はほとんどなくなっている。サブルーチンも引数を取ることができるし，関数も引数なしで使用できる。さらにいえば，サブルーチンに括弧をつけて実行してもそのまま通ってしまう。こうなるとサブルーチンは値を返せないだけで関数とあまり変わらない。関数のほうが概念的には上位にあるので，全体を関数だけに統一してもほぼプログラムは成立する。混在しているとまぎらわしいので，個人的には関数に統一することをおすすめしたいのだが，一部サブルーチンの記述が強制される局面もあるようで，なかなか一筋縄ではいかない。VBの本質ってのはこういうところにあるのかもしれない。

結局は好きに使い分けるしかない。

のようにすると，Label1のオブジェクトの文字の色が赤く変わる。従来のBASICにはなかった処理で，これは多用されることになるだろう。

オブジェクト名.プロパティ

と書くことで，そのオブジェクトのプロパティにアクセスできる。どんなプロパティがあってどういう風に変更できるのかはそれぞれのオブジェクトごとに違っている。

・メソッドを使う

オブジェクトに直接命令を送ることができる。なにができるかは，そのオブジェクトがどんなメソッドを用意しているかにもよって違ってくる。

```
PictureBox1.line (500,500) - (1500,2000),
Vbblue
```

などとやるとピクチャーボックスに線が引かれる。

なお，「PictureBox1」とかいうのはVBが自動でつけている名前というだけで，実際にはプロパティの「オブジェクト名」の部分を変更することで自由に変更できる。もっとも，プログラムをかなり書いたあとに変更するとコード部分の修正が面倒になるかもしれないが(コード部分までは自動的には変えてくれない)。

■とにかく実践編

理屈ばかりだと面白くないので，少しだけ実技を入れていこう。

プログラミングスタイルについてはVBらしくやるというのをあまり考えずにやっていく予定だ。とりあえず少しずつ慣れていくことが重要だろう。変数名のつけ方とか一応マイクロソフトおすすめ形式とかもあるのだが，それには準拠しない。

まず，VB5CCEを入手して起動してほしい。

起動すると例によってどんなタイプのプロジェクトを作るのかと聞いてくる。ここではコントロールグループを選択しておこう。私の場合，とりあえずコントロールグループを作ることにしている(面倒なこともあるのだが)。異論のある人は山ほどいそうなので強くはすすめないが，これが私のスタイルだ(CCEだし正解だとは思うけど)。

ここでActiveXコントロールを作成すると，プログラムの実行時にEXEファイルのプロジェクトを追加しなくてはならない。EXEを選択すると，本当にEXEだけなので，ほかのものを追加するときに面倒だ。コントロールグループを選択すると，ActiveXコントロールとEXEの両方を作るようになる。コントロールが必要ないときにはEXEの部分だけいじってればいいし，EXEで作っておいてActiveXにしたくなったらちょっと面倒だが，ユーザーコントロールで作っておけば，ActiveXを新規に作って，その上にコントロールを置くだけでいい。もちろんパフォーマンスなどは無視しているけど，それを気にするなら，どれでもだいたい同じ手順になる。コードを移し替える手間はEXEよりむしろ少ないだろう。

ユーザーコントロールを作るときには，最初だけEXEと違う手順が必要だ。ユーザーコントロールをだいたいデザインしたら，一度閉じて，フォームのほうのデザイナを立ち上げる。コントロールにいま作った新しいユーザーコントロールが加わっているので，それを選択して適当な大きさでメインフォームに張りつけよう。

ユーザーコントロールはあくまでも部品なので，単体では実行できないのだ。

■コントロールを作る

「ボタンなどの部品をコントロールと呼ぶ」と書いたのだが，実際には「なんでもかまわない」。ひとつのアプリケーションに近いようなモノを作ってコントロールにしても全然OKだし，私はむしろその方針でやっている。その過程で部品になるコントロールを作っていくのが正しいやり方だろう。

まず初歩編で「画像を読むコントロール」だ。

単なるテキストボックスだが，ダブルクリックされるとファイルを読み込み，最初の1行を表示する。

ざざっといってみよう。

まず，VB5CCEを起動し，コントロールグループを選び，右上のプロジェクトグループという部分からフォーム(Form1)を選んで編集し(ダブルクリック)，左のバーのコントロールのいちばん下にある変なアイコンをダブルクリックしてユーザーコントロールを張りつける。あとは右に戻ってユーザーコントロールをダブルクリック，ユーザーコントロールの編集モードにしたらあとはそのままででいい。わかりにくいけど，一度覚えれば済むので，ぱぱっとやってみよう。

図2　プロジェクトグループを選び

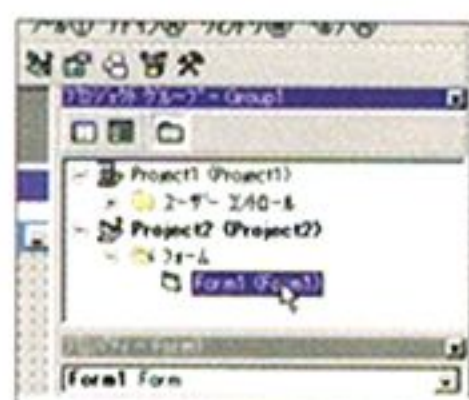



図3　コンポーネントを追加し

図4　ダブルクリックの処理を加える

さて，いきなりだが，ここでコモンダイアログというのを使う。ファイル関係を扱うときに必須の便利なものなので，真っ先に覚えること。

メニューのプロジェクトから[コンポーネント]というのを開く。リストの中にMicrosoft Common Dialog～というのがあるので，それをチェックして読み込む。すると新しいパーツとしてコモンダイアログというのが使えるようになる。

これが左のバーに追加されるのでダブルクリックだ。ふう。さて，次にテキストボックス(ABCとか書かれた奴)を持ってきて適当に配置する。

次にテキストボックスをダブルクリック。これでコードウィンドウが立ち上がる。デフォルトでは，

```
Text1_Change()
```

というサブルーチンになっているので，コードウィンドウの上のリストボックスからDblclickを選んで新しいサブルーチンを作る。中身は，

```
Private Sub Text1_DblClick()
Dim file1, file2, tmp As String
Dim i, j, k, fn As Long

file1 = CommonDialog1.filename
CommonDialog1.ShowOpen
file2 = CommonDialog1.filename
If file2 <> file1 Then
  Open file2 For Input As #1
```

プログラムを始める前に
COLUMN

初めてこういったツールに触るという人のためにちょっとひと言。

まず，最初にディレクトリを作っておくこと。

VBはほっておくと似たようなファイル名のファイルを作るのでファイルの管理がしにくい。プログラマが意図的に作成するファイル以外にいつのまにかできているファイルが多いので，状況を把握しづらいのだ。ちゃんとしたものを作るときにはちゃんとやればいいのだが，手軽になにか作りたいというときにはいちいちデフォルトネームを変更していくのも面倒になってくる(この辺は使っていればすぐにわかるだろう)。

バージョン管理などでのトラブルもあるので，なにか作るときには必ず新しいデータディレクトリを作るようにしよう。プログラムを一部変更して前のバージョンも残したいというときも同じである。まずディレクトリごと複製してから作業をすると混乱しなくていいだろう。

```
        Line Input #1, tmp
        Text1.Text = tmp
        Close #1
    End If
    End Sub
```

といった感じになる。

ファイル処理のいちばん基本のかたちがこれだと思ってもらってもいい。ツールなどを作る際にもっとも使用頻度が高く，使いものになる処理がファイル処理である。BASICは文字列の扱いなどが楽なので，基本を覚えればちょっとしたことがすごく便利になってくる。

VBだけでもファイル選択などはできるのだが，ファイル名を選ぶときはコモンダイアログと決めておけばかなり楽ができるので最初に覚えておこう。

なお，このコントロールは1行表示だけなので，実はあまり使いものにはならない。1行目が空白だったりするとなにも出ないし，そういうファイルは結構多いのだ。ループを組めば複数行表示も可能だ。なお，その場合は，

```
    Text1.Multiline=True
```

とかいったものを加えておこう。

非常に簡単だが，一応これでも「ユーザーコントロールを作った」ことになる。VBにはもっともっといろんなものがついてくるし，さらに無限ともいえるくらいに拡張ができる。コンポーネントの読み込み時にあったものは，使い方さえわかれば(使うのは楽ではないだろうが)大半はVBでも使える部品なのだ。

■変なコントロールを作る

コントロールというのはさっきも書いたように，プログラムで使える「部品」だ。それぞれが表示される形を持っているし，指定すれば一定の動きをするので，中身のことを知らなくても使い方さえわかっていればいい。独立していて，なにかあるとイベントを起こしてメインプログラムで処理できて，メインプログラムから指示をすると勝手に動いてくれる……。

という風に結構オブジェクト指向なわけだが，この辺がよくわからない人は別に気にしなくてもいい。おいおい説明していくので大丈夫。

「ゲームはやらない」

と明言したのだが，コントロールは可能性としては面白いものを持っているので少し触れておこう。

ゲームというものがオブジェクト指向なものになるというのはOh!Xで10年くらい前にやっていたことだが，時代は着実にそう流れている。現状のゲーム機だとメモリ足りないんでC++は使いにくいらしいのだが，「次世代」では，間違いなく主流になる。

さて，ここでボタンの代わりに敵キャラクターを配置して，それぞれがプログラムされていて，弾が当たったとかイベントを返してきて……という展開を想像してほしい。そういうコントロールを揃えれば，VBはそのままゲームエディタになる。メインプログラム側に多少のマネージャは必要なんだけど，別に嘘ではない。

基礎実験をしてみよう。

まずキャラクターだ。GIFの256色で作って，マスク部分は透過GIFにして張りつける。OKだ。コントロールを張りつけて実行する……がマスク部分が透明にならない。いや，なってるんだけど，フォームの色が出てきて結果的に透過されていない。おいぃ，ビルぅ，なに考えてんだよぉ？

ここで四角形以外のものを出すにはShapeというのを使うしかない。フォームを透過色にするかどうかはShapeと文字でしか有効にならないからだ(まあ，文字でもいいんだけど)。Windowsの世界では四角いもの以外は扱ってはいけない。というか，あらゆるものが矩形単位で処理される。注意しよう(最近，DirectXをWindowsGDIに換えるような次世代GDI仕様が発表された。席上でのひと言「ウィンドウはなにも四角いものだと決まっているわけではない」……。どの口がいってんだぁ？　もしかして，本当につい最近それに気づいたのかもしれないんだが。彼らの常識は常人に理解できないことはご承知のとおりだ)。

んで，とりあえず，Shapeで円を出す。VB6CCEでは透過色がPictureBoxにも適用されていますようにと祈りを込めつつ，パーツを配置。

次にコードを書く。

コードウィンドウを開いたら，黙って先頭部分に，

```
    Option Explicit
```

と入力する。理不尽な災害から身を守ってくれるありがたいおまじないだ。

さて，用途からすると，このオブジェクトは自分で動いてもらわないといけない。位置指定は通常，ウィンドウの端からの距離をTop，Leftという値で指定する。

```
    Shape1.Top＝100
```

と書けば，100の値の高さに設定される。……のだが，自分自身を指定する方法がない。ユーザーコントロールの中でユーザーコントロール自身は指定できない。ほかの言語でありがちな，

```
    this.top＝
```

とかいうものが存在しないので，やや面倒な処理が必要だ。わからなければ親に教えてもらうことになる。親というのは張り込まれるFormのことだ。

Object型変数を使って，オブジェクトの型を受け渡す(そのまんまだな)。オブジェクトとはボタンなどのコントロールを含むいろんなものだ。親は子供を知っているが，通常，子供は親を見れない。そこで親との交信のために，親の名前も渡してやろう。

図5　自走コントロールの基本設計

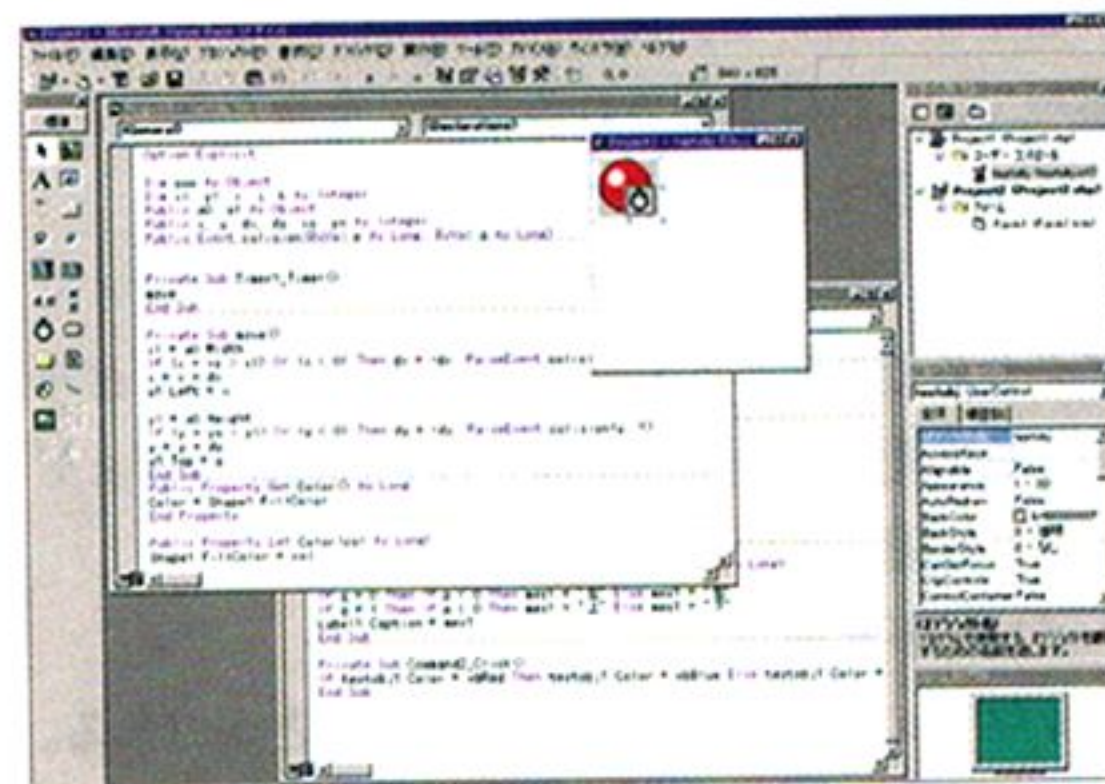
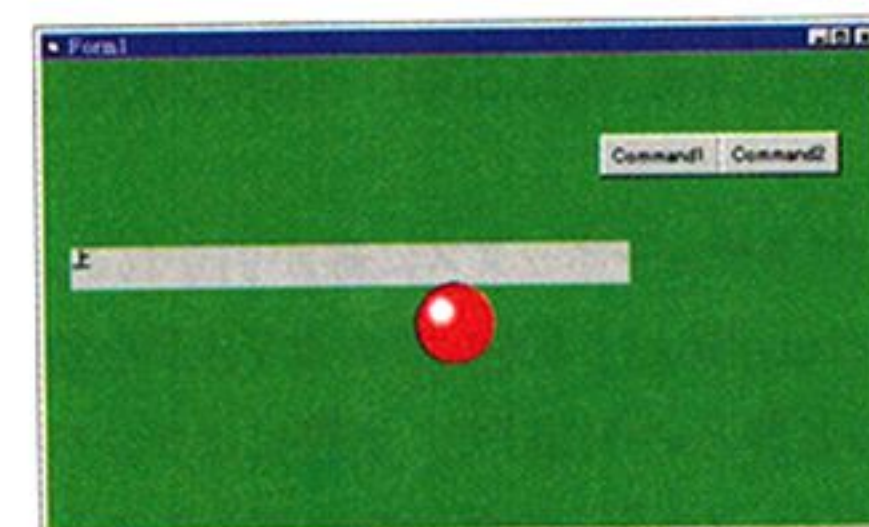



図6　ボールが自分で動いている

で，動かす。動いた。反射する位置が対称になってない気もするが，変なことをしたつもりはないので(してたっけ？)，気にしないようにしよう。VBとはこういうものなのだ(悟り)。

ついでなので，オブジェクトからイベントを発生して，処理を駆動できるようにしよう。メインプログラム側から，オブジェクトの状態を読み取ったり変更したりもしてみよう。

イベントを起こすにはRaiseEventというのを使う。なんじゃそりゃと思う人もいるだろう。最初はVBでなにができるのかが全然わからないと思うので，ヘルプファイルでいろいろ探してみよう。使いやすいとはいえないが，ヘルプを駆使できないとVBは使えない。見ればいろいろ載っているのだ。好きな名前のイベントが作れて，普通のボタンなどのイベントと同じようにBASICに渡されるので，同じように処理すればいい。

オブジェクトの状態はプロパティで変えるか，内部変数を直接参照する。プロパティはご覧のように設定する。この辺は理屈ではない。

よくわからない人はわからないかもしれないが，これでひととおりのことはできたことになる。あとは透過色さえ対応してくれて，実行速度が激速になれば結構面白いことができるだろう。うむうむ。

Windows GDIがDirectX化されて，マシンが爆速になったらまた掘り返すことにしよう。現状でも，Win32API呼び出しなどで十分に高速なモノを作れる。DirectXも使おうと思えば使えなくはない。が，指定が煩雑である。VBの範囲内だけでやれることをやっていくのが基本だろう(でもマイクロネットのDirect3Dライブラリなどは面白そうだなあ……)。

Future BASICによるMacintoshプログラム制作

古籏一浩/Furuhata Kazuhiro

プログラミングなどからはもっとも遠い機種と思われているMacintosh。しかし，やはりそこにも開発環境は存在する。ここでは手軽なツールとしてFuture BASICを紹介しよう。ちょっとしたものが自分で作れると応用範囲も大きく違ってくる。

■Macintoshという道具

Macintoshは道具だといわれることがある。手の感覚にフィットしたマウス，そしてその動き。直感的にわかりやすく，習得までの時間が短く人に優しいコンピュータ。シェアが小さいとはいえ，多くの数，種類のソフトが発売され，お店に行って自分にあったソフトを購入してくれば，ほとんど用は足りてしまう。

しかし用は足りるが「もの足りない」と感じたことはないだろうか？　使っているソフトに不満はないか？　自分がほしいソフトが開発され発売されるか？　それは本当に自分が望む機能と値段か？　使いものにならないものに大金を出してしまわないだろうか？　自分の遊びたいゲーム，本当に遊んでみたいというゲームはあるか？　自分が望むツールはあるか？

現状で十分に満足しているのであれば特に「プログラムを組む」必要はないだろう。シェアが低いといわれるMacだが，それは与えられたものを使い，経営不振のApple社に不満をいって，解決してくれるのを，口をあんぐり開けて待っている，そんな結果ではないだろうか？(現状の日本の景気と似たり寄ったりなのは偶然だろうか)　待つだけなら誰にだってできる。重要なのは，待つだけでなく「自分で作り上げていく力」，そして「自分の目指す未来に躍進する力」ではないだろうか。

「自分で作り上げていく力」……それは技術力だろう。技術力がないからなにもできない，というのは大きな間違いだと思う。大事なのは技術ではなく「自分の目指す未来に躍進する力」，つまり「情熱」だ。情熱なくしては，どんなに凄い技術を持っていても「無用の長物」だ。技術は後からでもついてくる。

Apple社が潰れたってどうということはない。そうなったら，自分でOSを作ってしまえばよいしアプリケーションも自分で書けばよい。

「そんなことできるわけない」

そう思っている間は本当に「できない」。いまはできなくてもあきらめては駄目だ。ソフトウェアの多くは技術の積み重ねとアイデアの積み重ねだ。どんなに高度なものでもアイデアと時間さえあれば作れるはずだ。要するにあきらめなければいい。

そして，

「なければ自分で作ればよい」

のだ。MZユーザーもXユーザーも，そうして独自の世界観を作り上げてきたはずだ。MZ，Xユーザーにできて Macユーザーにできないはずがない。ましてや昔と比べ開発環境にも恵まれている。作れないのは，「作れないと思っている」からだ。作る気になれば，あとは進むだけだ。「千里の道も一歩から」始まるのだ。

■Macintoshでプログラムを組む

Macintoshでプログラムを組むことは決して特別なことではない。結局のところ，Macintoshもコンピュータなので，それなりのプログラム言語と開発環境は用意されている。しかし，Macintoshに関しては情報が少ない[*1]ためか，Macintoshで開発することは奇異に見られることがある。C言語もあるしPASCAL，BASIC，FORTRANなど，意外と多くの言語が用意されている。

またMacintoshでプログラムを作成しようとした場合「Inside Macintosh(インサイドマック)」が絶対必要だ，といわれることがある。このインサイドマックがなにものかというとMacintoshで扱える関数呼び出しについて記述した本のことを示す。現在は本だけでなくCD-ROM，Web上でも使用することができる。CD-ROMとWebは英語のみ存在しているので英語がそこそこわかるのであれば特に高いお金を出して購入することもない。

ほかにも，デバッガであるMacsBugの本，リソースを編集するためのResEdit[*2]の本などがある。できればResEditの本は1冊購入し手元に置いておくほうがよい。

＊1　NIFTY SERVEのFMACPRO，インターネット上のホームページ，fj.sys.mac.programming，そして書籍などがいくつか出ている。書籍は新しいものを購入したほうがいい。1年経過すると状況が変わっているのは業界の常だからだ。またMacintoshの場合，アプリケーションのバージョンアップよりもOSのバージョンアップのほうが早いことが多い。

＊2　Macintoshで扱うプログラムに関する部品製作管理を行うようなもの。

■どのような開発環境があるか

いくつかの言語があると書いたが，現状で選択できるものでメジャーどころ(？)といったら以下のようなものがある。それぞれについて，簡単に書いておく。

●CodeWarrior

現在C言語で開発を行うとしたら，この「Code Warrior(コードウォーリア)」だろう。統合開発環境であり，多くのプログラムはCode Warriorによって開発されている。対抗馬もあったがすでに撤退してしまいCodeWarriorだけが残っている感じだ。コードウォーリアはMacintoshだけでなくWindowsなど複数のプラットフォームの開発も行うことができる。C言語を習得しているならばCode Warriorを選択するのがよいだろう。学生用としてアカデミックパックも用意されている。

現在の最新バージョンはCode Warrior Pro 3。発売元はメトロワークスだ。付録CD-ROMには体験版ともいうべきCodeWarrier Liteが入っているので，C言語に馴染みのある人は触ってみるのもいいだろう。

●Prograph CPX

「絵を描くようにプログラミングできる」これがPrograph CPXの売り文句である。確かに絵を描くようにプログラミングできるのだが，オブジェクト指向に慣れていないと戸惑うかもしれない。

プログラムの作成は，オブジェクトを画面に配置し，そのオブジェクト間を線で結ぶという仕組みになっている。配置したオブジェクトが複雑なレイアウトになってしまう場合でも，オブジェクトを選択してひとつにまとめることができるためプログラムの見通しもいくらかよくなる。ほとんどは配置，結線のくり返しともいえる。

●RealBasic

Macintosh版VisualBasicといった感じだろうか。手軽にレイアウト/インタフェイスが作成でき，Macintosh上で動作するだけでなくJavaコードも吐き出せる。今後のMacintoshのプログラム環境の最有望株といえる。雑誌などでも記事が連載されている。また，サンプルが掲載されているホ

ームページもあるので参考にするとよいだろう。

●Future BASIC II日本語版

Macintoshの開発環境で珍しく「日本語版」として出ているBASICコンパイラである。BASICといっても，Macintoshの深い部分まで踏み込んだプログラムを作成することができる。BASICではあるが，ダイレクトに68000のマシン語を埋め込んで利用することが可能だ。深い部分まで踏み込める上に，手軽にアプリケーションが作成できるところがFuture BASIC II日本語版の最大のポイントだろう。ただし，Future BASIC IIはPowerPCコードを吐き出せないという欠点もある(バージョン3ではpowerPCコードも吐き出せるとのこと)。

今回，使うのはこのFuture BASIC II日本語版(以下Future BASIC)である。手軽で深いところまで進める，それが選択理由だ。

■BASICとは？

BASICはコンピュータ言語の一種だ。もともとはFORTRANよりも手軽に演算を行うために作成されたようだ。実際いちばん最初のBASICの命令を見ると行列演算などが手軽に処理できるようになっている。これが，いつの間にか「初心者にも使えるやさしいコンピュータ言語」といわれるようになり，パーソナルコンピュータの普及とともに原形をとどめないほどに進化(？)発展し一大ブームを築いた。1980年代前半は「コンピュータが使える＝BASICでプログラムが作れる」という図式に近いものがあった。

しかし作成されるプログラムが次第に巨大になっていくにしたがい，BASICの限界が見えるようになった。そして大規模開発に向かないBASICは衰退し，開発言語の主流はC言語へと移行した。BASICは時代から取り残されたかのように思われたが，VisualBasicなどの普及により再び多くの場所で利用されるようになった。これは従来のBASICの不備な点を改良し，手軽に利用できるようにしたためである。このように従来のBASICの不備な点を解消したBASICもある。もちろんFuture BASICも従来のBASICの問題点をいくつか解消している。

よく初心者の質問に「開発言語はなにがよいですか？」というのがある。Macintoshの場合，おおよそ答えは決まっている。「Code Warriorにしましょう」といわれるのだ。つまりC言語で作成しなさい，ということだ。作成対象にもよるが，とにかくC言語をすすめてしまうのには難があるのではないか。

BASICにはBASICなりのよさがある。それは「手軽さ」だ。C言語で20〜30行必要な部分もBASICなら2，3行ですんでしまう。当然プログラムもわかりやすく，見通しがよくなる。BASICでMacintoshの基礎的な部分を学んでC言語にステップアップすればよいのだ。特にFuture BASICはプログラムの書き方がC言語に近いものがあるので勉強しておいても損はないはずだ。「急がば回れ」ということだ。

■プログラムの組み立て方

いきなりFuture BASICを買ってきてプログラムを作ろうと思っても，そう簡単にできるものではない。マニュアルには，練習用のものがあるので入力しつつ勉強するのも手である。練習よりも実用的なものを作成したいのであれば，ハンドブックマニュアルのサンプルを参考にプログラムを改造，改良してみるといい。

問題なのは「オリジナルプログラム」を作成する場合だ。とりあえず作成するプログラムの概要を紙にでも書き出して画面のレイアウトなどを行ってみるとよいだろう。とりあえず紙の上でもイメージがつかめれば十分だ(頭の中でできるなら，それにこしたことはない)。

次にどんな処理をしなければならないか書いてみるとよい。ウィンドウを開く，時刻を表示する，メニューを選択するなど，単純な部分を列記してみる。列記したら今度はFuture BASICのリファレンスマニュアルを開いてみよう。たくさんの命令が用意されている。列記したものに対応する命令をマニュアルから探してみるとよい。大半は探すことができないか命令自体存在しない。この場合は，いくつかの命令を組み合わせて，列記した処理を実現しなければならない。こうなったら，さらに細かく分けて書いてみる。書いたら先ほどと同様にマニュアルを開いて対応する命令などに置き換えていく。

こうやって書くとプログラムは面倒と思うだろうか？　実際面倒だ。だから，自分の作ったプログラムが正しく動作した場合は，それだけ感動も大きいのだ。

それでは，次にFuture BASICでの約束ごとについて書いていこう。

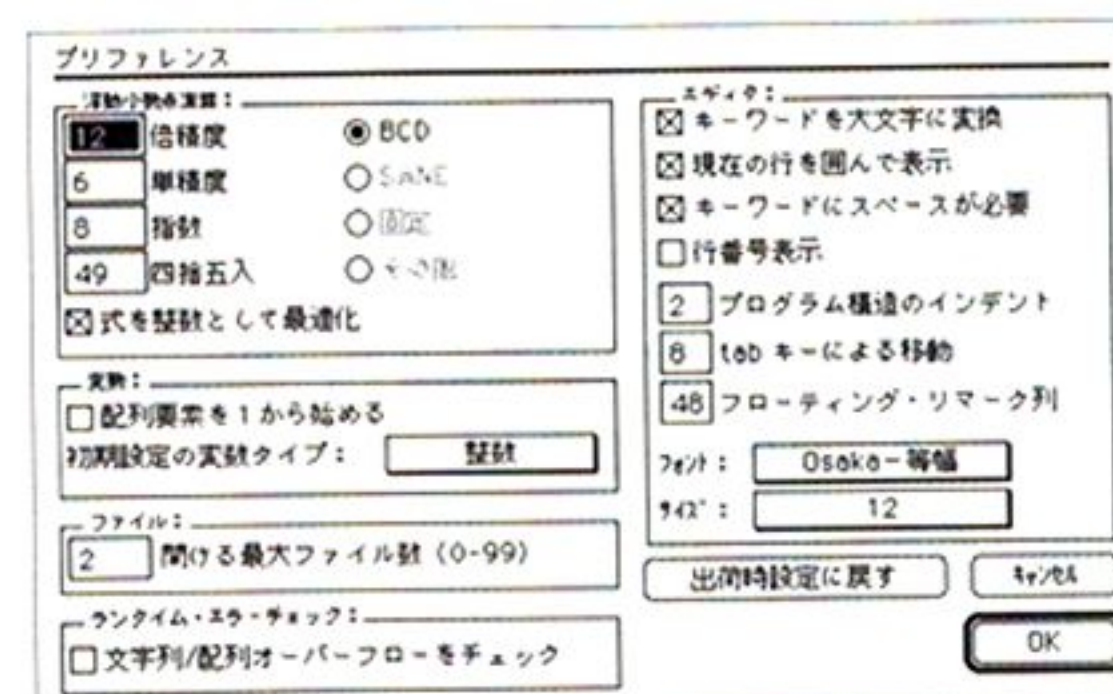
■変数について

変数とは，特定のものが格納できる箱のようなものだ。この箱に文字や数値を入れたり出したり演算したりする。この箱には名前をつけることができる。名前は最大240文字までつけることができるが，実際に判別されるのは先頭から15文字目までである。長い変数名を使うときは注意したほうがよい。

またなにもコンパイルオプションを指定しない場合[*3]「変数名の大文字と小文字は区別」される。つまり，a＝1とA＝2とは別の変数ということになる。単純だが十分気をつける必要がある。

Future BASICで扱える変数の種類には以下のものがある。

- 整数型(変数名の末尾に%を付加)
- ロング整数型(変数名の末尾に&を付加)
- 単精度型(変数名の末尾に!を付加)
- 倍精度型(変数名の末尾に#を付加)
- SANE拡張型(変数名の先頭に#を付加)
- 文字列型(変数名の末尾に$を付加)

変数名の末尾になにもつけない場合は，プリファレンスの設定によってコンパイラが自動的に判断する。プリファレンスで設定できるのは「整数」「ロング整数」「単精度」「倍精度」となっている。初期状態では「整数」になっている。

プリファレンスではFuture BASICの基本動作に関する各種の設定が行える

・整数型

整数型の変数は－32768〜32767までの数値しか扱うことができない。数値だけしか入らず文字列は入れることができない。a%＝123は問題ないがa%＝"Oh!X"とするとエラーになってしまう。また小数も扱うことができず小数点以下は代入時に切り捨てられてしまう。

・ロング整数型

ロング整数型の変数は－2147483648〜2147483647までの数値を扱うことができる。整数型と同様に文字列を入れることはできない。

・単精度/倍精度型

単精度型と倍精度型は，浮動小数点演算を行う場合に利用する。演算はBCD(Binary Code Decimal：2進化10進数)で行われるため演算誤差は極めて少ないが，演算には時間がかかる。演算精度はプリファレンスで設定することができる。この型も文字列を入れることはできない。

・SANE拡張型

SANE拡張型はApple社が用意している浮動小数点演算パッケージの書式にあわせた変数形式である。ほとんど使うことはないが，高速に浮動小数点演算を行いたい場合に利用するとよい。かなりの速度向上が見込める。日本の販売代理店モード社のページ(http://www.mode.co.jp/)にサンプルがあるので参考にするとよいだろう。

・**文字列型**

文字列型は255文字以内の文字列を格納することができる。文字列を代入する場合は数値型の変数とは異なり"(ダブルクオーテーション)で代入する文字列を囲む。たとえばa$にoh!mzという文字列を代入するにはa$ = "oh!mz"のようにする。

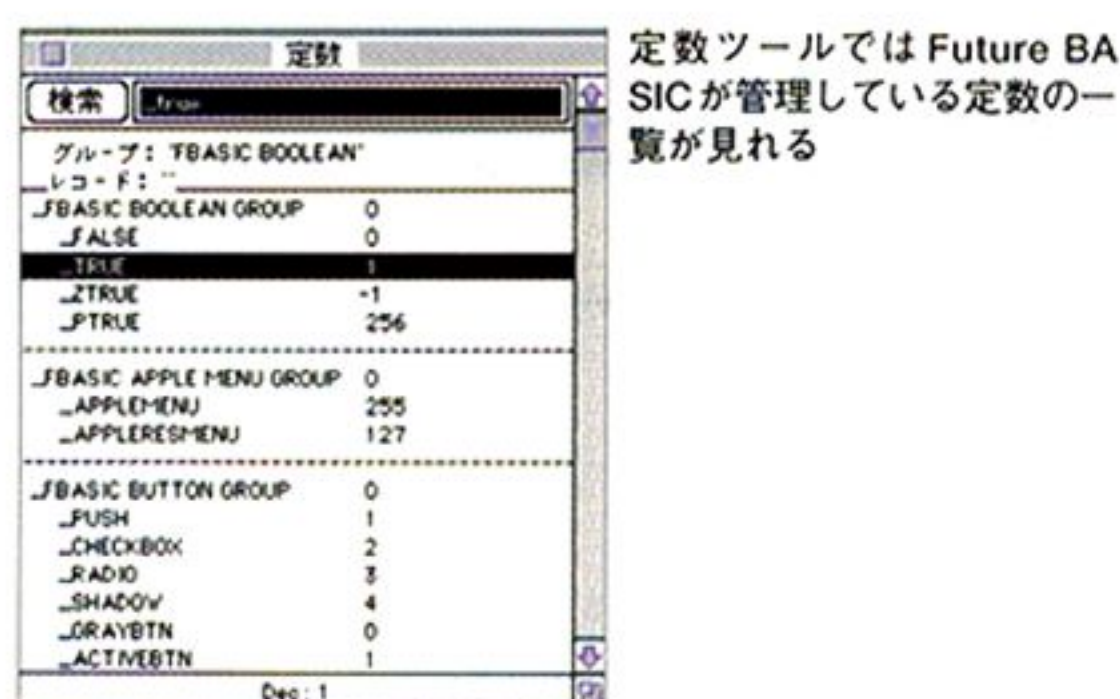
＊3　大文字，小文字を区別させないようにするには「COMPILE 0,_caseInsensitive」をプログラムの先頭に記述する。

■定数について

定数とは，あらかじめ定められている値のことだ。0や1なども定数だ。しかし，0や1ではいったい「なにを表しているか」がわからない。プログラムを見やすくするための方法として，定数名を定義しておくことができる。定数名は変数名と一緒だが，先頭に_(アンダーバー)がつく，プログラム途中で値を変更できない，あらかじめ定義されているものがある，大文字と小文字を判別しない，などいくつかの点で異なっている。

定数名を定義するには以下のようにする。

```
_定数名 = 値
```

値に文字や文字列は使用できない。_alpha = ASC("A")のような命令を利用した代入もできない。純粋に数値だけが使用できる。定数名がすでに定義されている場合は，定数名の再定義となりエラーになる。すでに定数名が存在するかどうかは，「ツール」メニューの「定数」を選択し表示される定数ウィンドウ上で検索することができる。すでに存在している定数名はプログラム内で書かなくても，そのまま利用することができる。

定数ツールではFuture BASICが管理している定数の一覧が見れる

また，定数名を使うことにより，プログラムに変更が入った場合でも手軽に修正することができる。たとえば以下の2種類のプログラムがあるとする。

●その1

```
a = 123
b = 123
if a = 123 then beep
```

●その2

```
_atai = 123
a = _atai
b = _atai
if a = _atai then beep
```

ここでプログラムの仕様が一部変更され123という値が456になったとする。その1の場合の変更個所は3カ所である。これに対しその2では先頭の1カ所だけ変更すれば終わりだ。3カ所程度ならば別段問題ないが，このような部分が，プログラム中に数十カ所あったとすれば，修正に手間がかかるしプログラムの誤動作(バグと呼ばれる)にもつながる。

■関数について

関数は「ある値を与えたら値を返す」処理の塊のことだ。与える値，返す値は数値である必要はなく文字列でも構わない。また必ず値を与える必要もなく，返す必要もない。なにも値を返さない関数もあれば，値を返す関数もある。この部分は作成者に一任されており作る人の自由だ。

昔のBASICを知っているのであれば「サブルーチン」といったほうがわかりやすいだろう。Future BASICの関数はサブルーチンよりもいくつか優れた点がある。まずローカル変数が扱えるということだ。ローカル変数というのは，関数内でしか利用することができない一時的な変数を示す。これに対してプログラム内であれば，どこでも使用できる変数をグローバル変数と呼ぶ。関数内で使用される変数はなにも指定しなくてもローカル変数になる。グローバル変数の場合は関数が定義されていない部分(以下に示すLOCAL FN～END FN以外の場所)で使用されていれば自動的にグローバル変数となる。

ローカル変数が扱えるとなにが便利だろうか？それは，どんな変数を使っても外部に影響を与えないので，プログラムパーツの隠蔽性が高くなり再利用性が高くなるということだ。関数の再利用性が高くなれば，別のプログラムを作るときにも，いま作成しているプログラム(関数)をそのまま使えるということだ。

関数を作成する場合はLOCAL FN～ENDFN命令を使う。この命令の書式は以下のようになる。

```
LOCAL FN 関数名(パラメータ1,パラメータ2,...パラメータn)
  :
  処　理
  :
END FN = 戻り値
```

戻り値やパラメータはなくてもかまわない。この場合は以下のようになる。

```
LOCAL FN 関数名
  :
  処　理
  :
END FN
```

関数名も変数名と同様に大文字，小文字を判別するので注意が必要だ。

作成した関数は，そのままでは実行されない。関数を実行する(呼び出す)には以下のようにする。

```
変数名=FN 関数名(パラメータ1,パラメータ2,……パラメータn)
```

戻り値がない場合は変数は不要だ。戻り値を入れる変数の型と関数の戻り値の型は一致している必要がある。一致していない場合はエラーになってしまう。

呼び出す場合に注意しなければいけないのは，呼び出す関数名は呼び出す側より先に定義されている必要がある点だ。つまり，呼び出される関数が呼び出すプログラムより先頭のほうにあればいい。たとえば以下の例1はエラーになるが例2は正しくコンパイルされる。

例1：

```
FN oh_mz
LOCAL FN oh_mz
  :
END FN
```

例2：

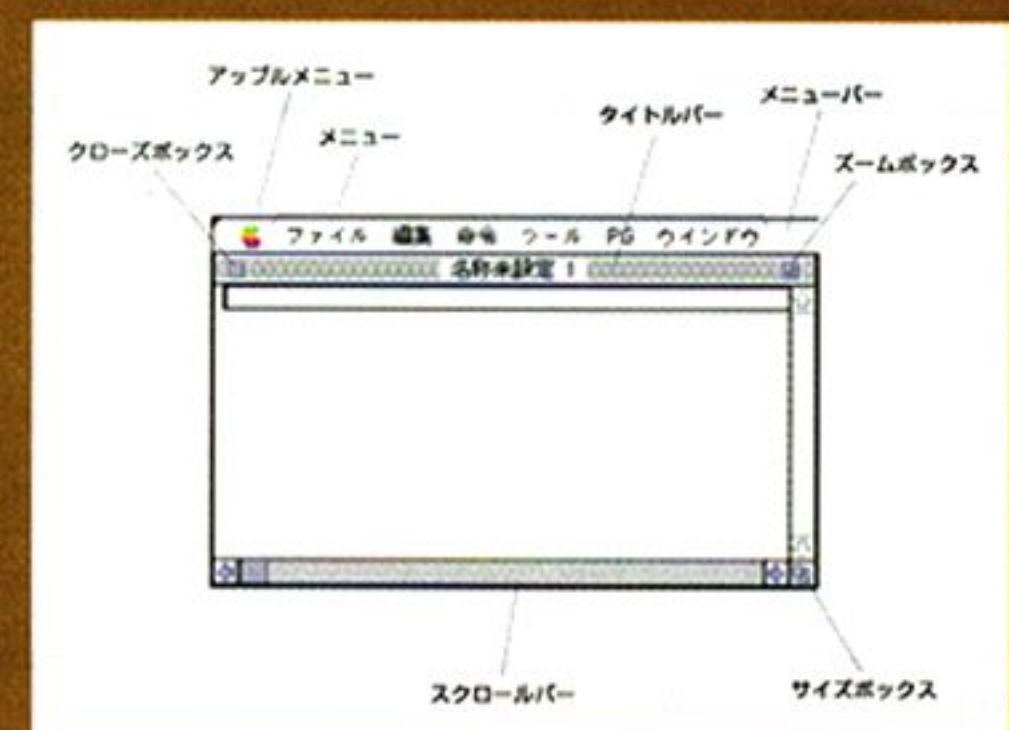
```
LOCAL FN oh_mz
  :
END FN
FN oh_mz
```

■時計を作る

今回作成するアプリケーションは，シンプルな「デジタル時計」である。ウィンドウに現在時刻を

ウィンドウパーツの名前について　COLUMN

Macintoshでプログラムを作成する場合，各場所の名称を覚えていなければ話にならない。各部名称は以下のとおり。

アップルメニュー
メニュー
クローズボックス
タイトルバー
メニューバー
ズームボックス
スクロールバー
サイズボックス

表示するだけだが，Macintoshのプログラミングのいくつかの重要部分を含んでいる。今回の時計に含まれるプログラム上のポイントは以下のとおりである。

1）ウィンドウを開く/閉じる
2）メニュー項目を選択，分岐処理させる
3）イベント

また作成するアプリケーションの概要は以下のとおりである。

1）起動後ウィンドウを開く
2）ウィンドウ上には1秒ごとに現在時刻を表示する
3）メニューバーの項目は「アップルメニュー」「ファイルメニュー」のみ
4）アバウト項目を選択するとアバウトウィンドウが開く
5）アバウトウィンドウにはアプリケーション名を表示しクリックされるまで待つ(クリック後はウィンドウを閉じる)
6）ファイルメニューは「終了」の1項目だけで選択するとアプリは終了する

これらは関数に分けて考えると楽だ。今回作成する関数は以下のものだ(カッコ内は関数名)。

1）アバウト画面の処理（printAbout）
2）メニューバー/メニュー項目の初期化（init Menu）
3）メニューの選択処理（selectMenu）
4）時計を表示する（printTime）

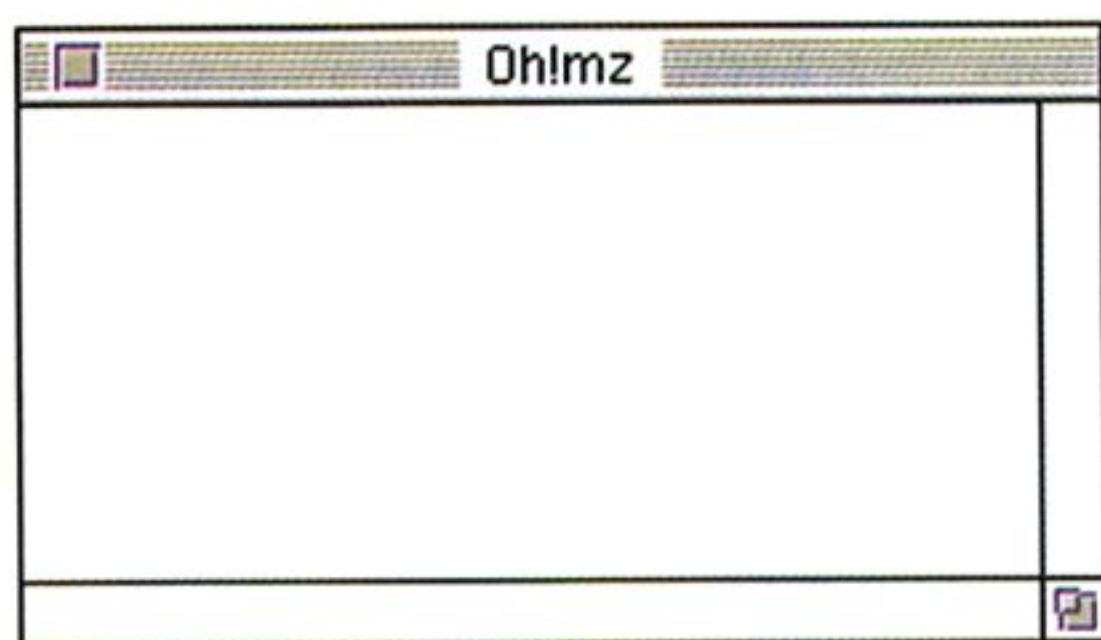
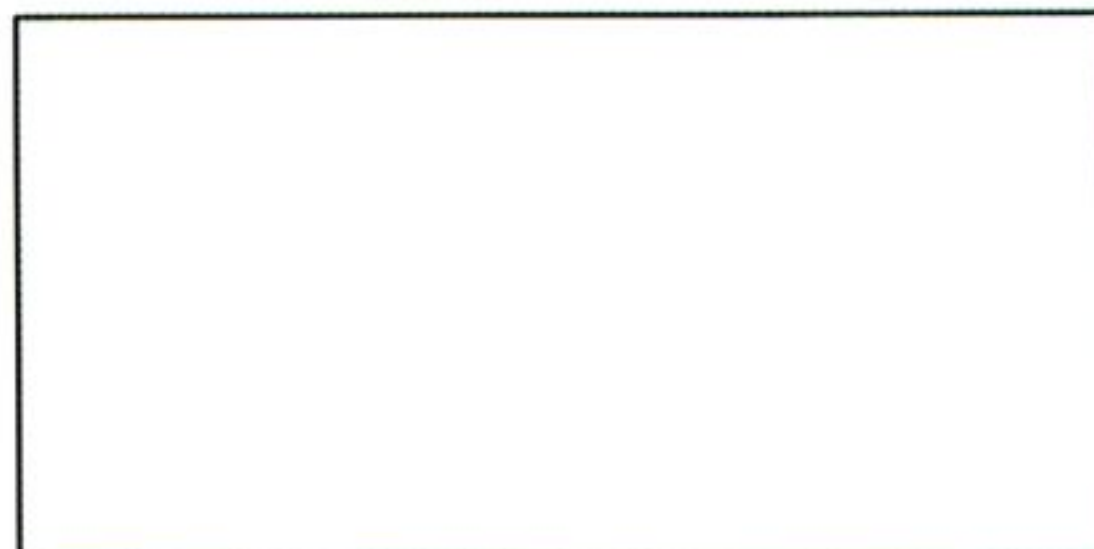
以下に作成するアプリケーションにおいて必要となる部分を説明する。また，それぞれに応じた短いサンプルも用意した。

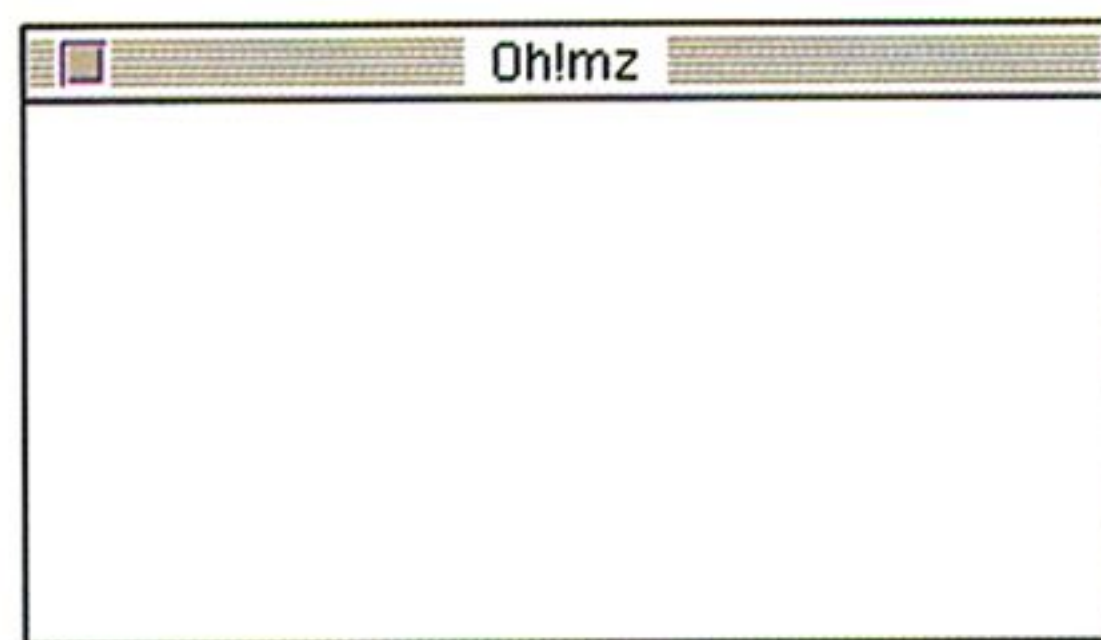
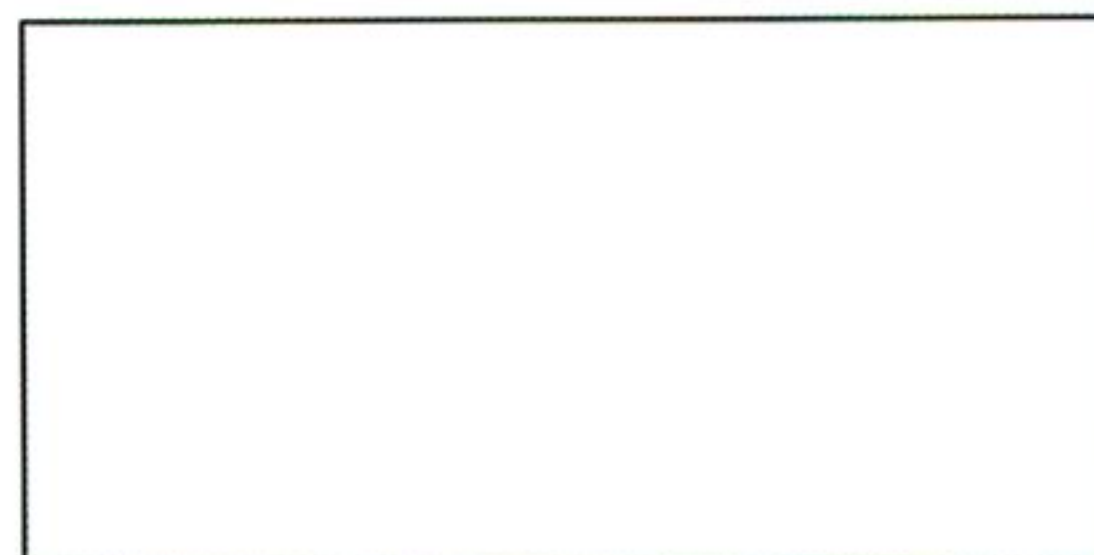
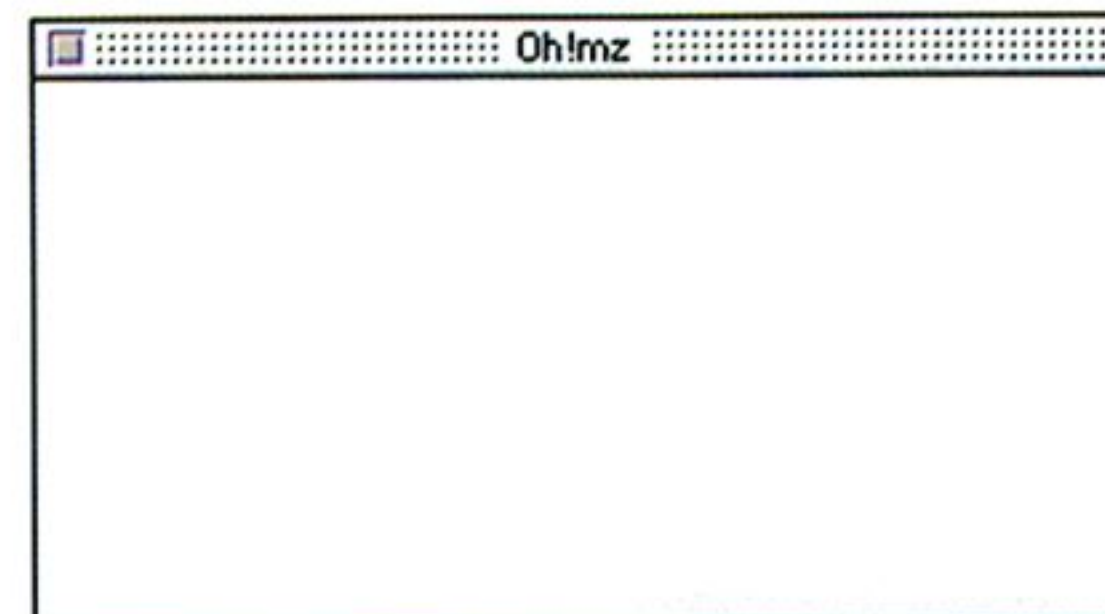
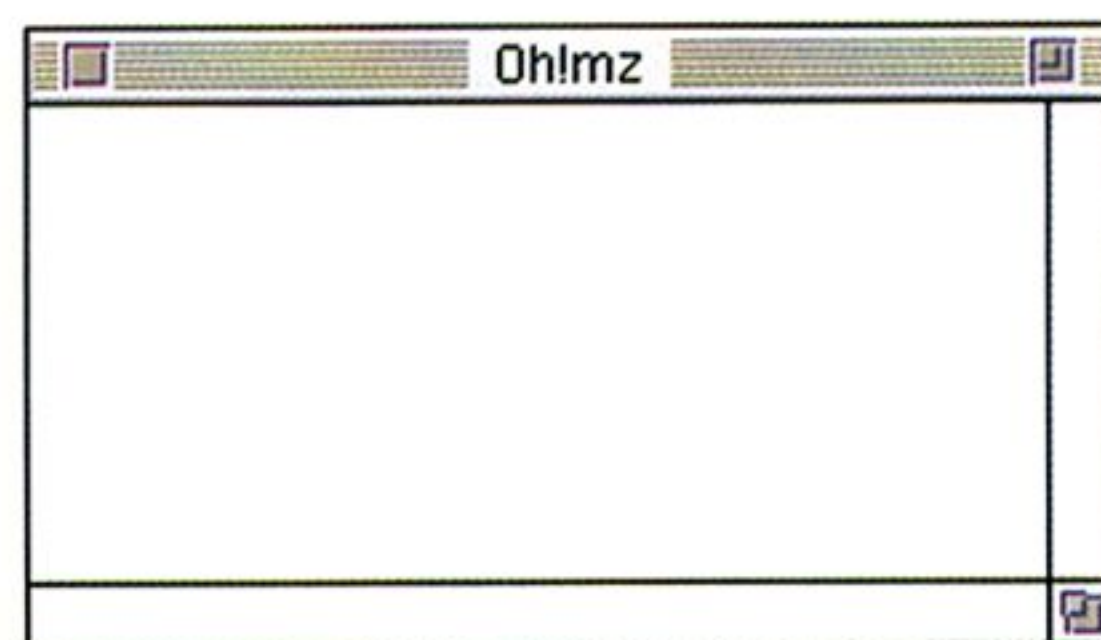
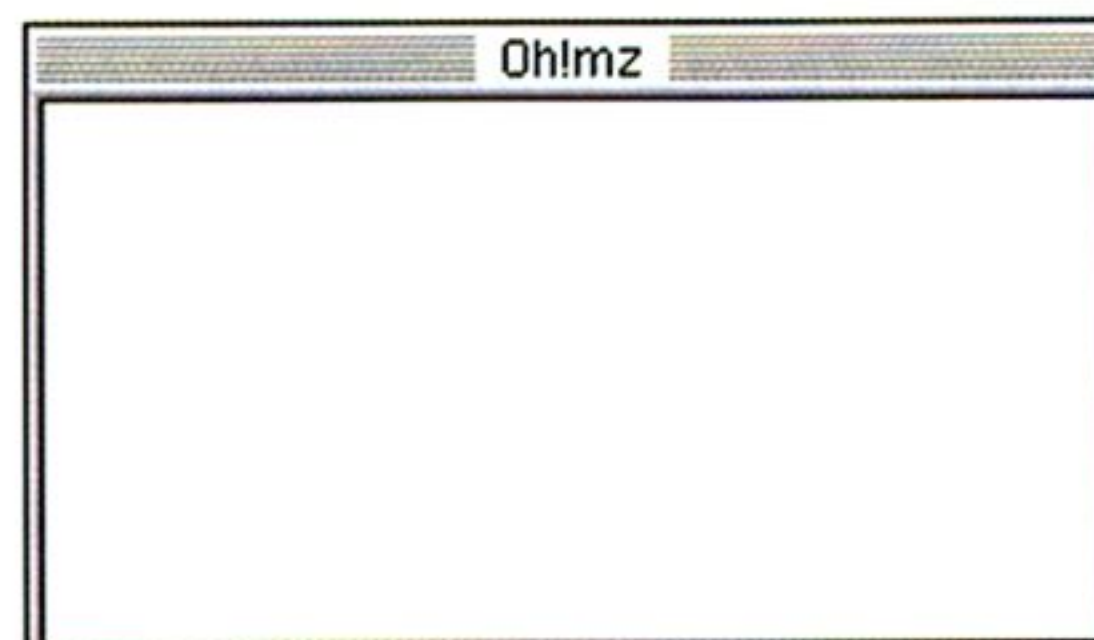
■ウィンドウ

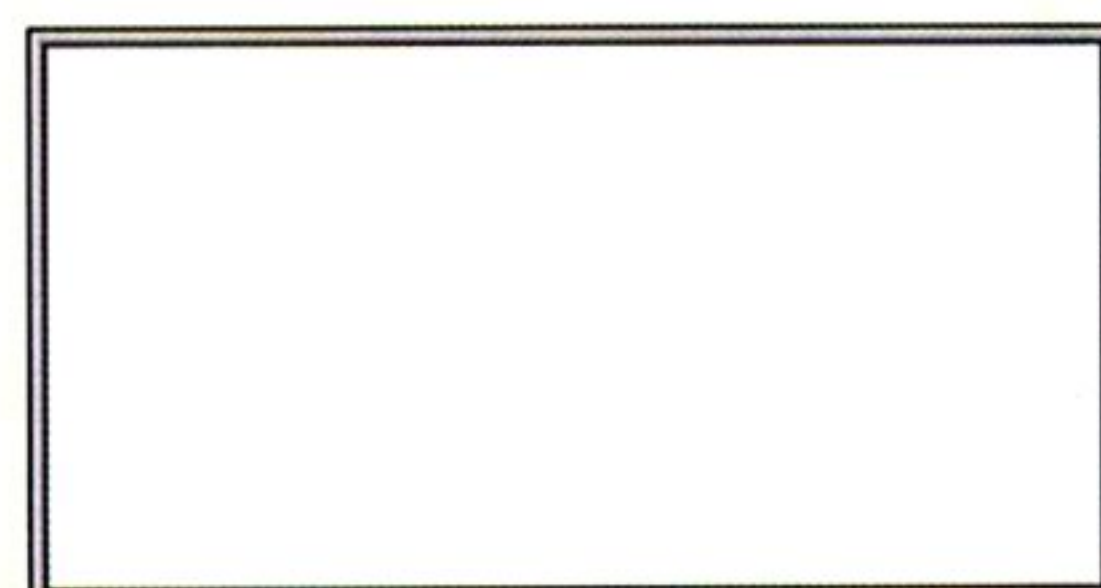
Macintoshのアプリケーションの多くは「ウィンドウ」を利用する。もちろんFuture BASICでも扱うことができる。ウィンドウを開く場合はWINDOW命令を使う。書式は以下のとおりだ。

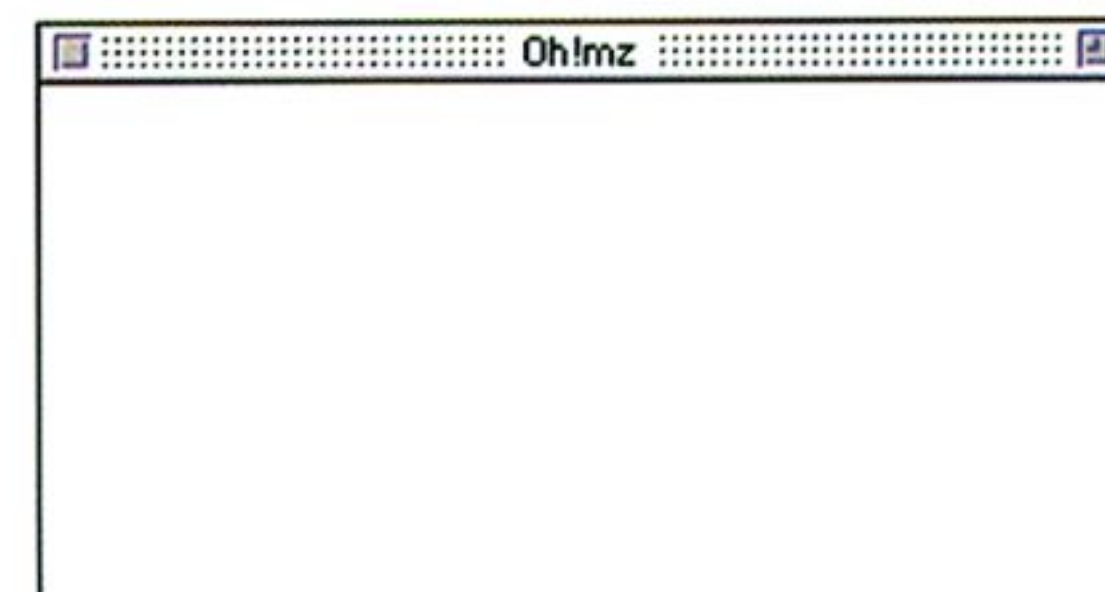
WINDOW [#]番号,ウィンドウタイトル,[(左端座標,上端座標)ー(右端座標,下端座標)],ウィンドウ形状

作成するウィンドウには「個別の番号」が必要だ。特に1から始まる必要はなく1～255までで

ウィンドウの種類いろいろ

Oh!mz
ドキュメントウィンドウ(_doc)

枠なしダイアログ(_dialogPlain)

Oh!mz
標準パレット(_WDEFbaseID_noGoAway)

Oh!mz
ドキュメントウィンドウ/リサイズボックスなし(_docNoGrow)

影付きダイアログ(_dialogShadow)

Oh!mz
クローズボックス付きパレット(_WDEFbaseID)

Oh!mz
ドキュメントウィンドウ/リサイズボックスあり(_docZoom)

Oh!mz
移動可能ダイアログ（_dialogMovable)

Oh!mz
横向きパレット(_WDEFbaseID_Wdefsidedrag)

枠付きダイアログ(_dialogFrame)

Oh!mz
角丸ウィンドウ(_docRound)

Oh!mz
リサイズボックス付きパレット(_WDEFbaseID_WdefhasZoom)

あれば自由に使用することができる。－1などのように負数を指定した場合は，新しいウィンドウは作成されずに該当する番号のウィンドウが表示されなくなる。表示されないだけでウィンドウがなくなるわけではない。再度該当する番号を指定すれば表示される。

ウィンドウタイトルはウィンドウのタイトルバー(いちばん上のバー)に表示する文字である。255文字以内ならば自由に名前をつけることができる。

ウィンドウの座標指定だが，画面の左上が座標(0，0)となっており右下にいくにしたがって座標は大きくなる。(10，10)－(330，250)と指定すれば左上(10，10)の座標から(330，250)までの範囲にウィンドウが表示される。

ウィンドウの(左端座標，上端座標)が(0，0)の場合は特殊な動作になる。(0，0)を指定した場合，作成されるウィンドウの座標は画面の中央に配置(センタリング)される。起動時の画面やアバウト画面など中央に表示したい場合に便利だ。

ウィンドウにはいくつか種類があり前ページのような形状が利用できる(カッコ内はFuture BASICで定義されているウィンドウ形状の定数名)。

また同時に開くことができるウィンドウの枚数は255枚となっている。これだけあれば十分だろう。さすがに255枚開くと，ウィンドウ周りの動作速度は低下してしまう。

Future BASIC上からプログラムを実行する場合は関係ないのだが，作成したプログラムをアプリケーション化した場合，起動時に1枚勝手にウィンドウが作成されてしまう。多くの場合，この起動時に表示されるウィンドウは必要ないだろう。このウィンドウを表示しないようにするには以下のようにする。

WINDOW OFF

この命令を書いておけば起動時に勝手にウィンドウが作成され表示されることはなくなる。

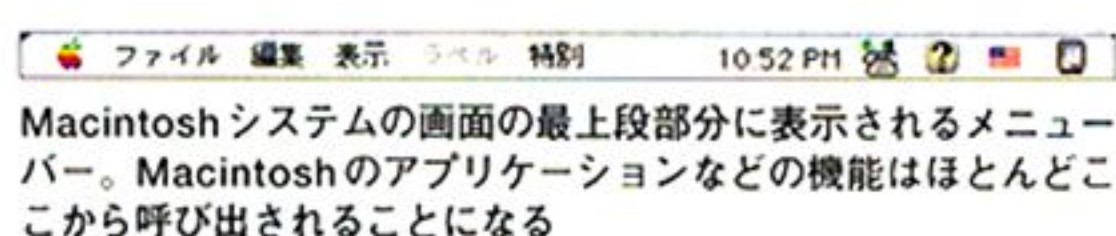
以下のサンプルはウィンドウを1枚開くものである。マウスのボタンを押すと終了する。

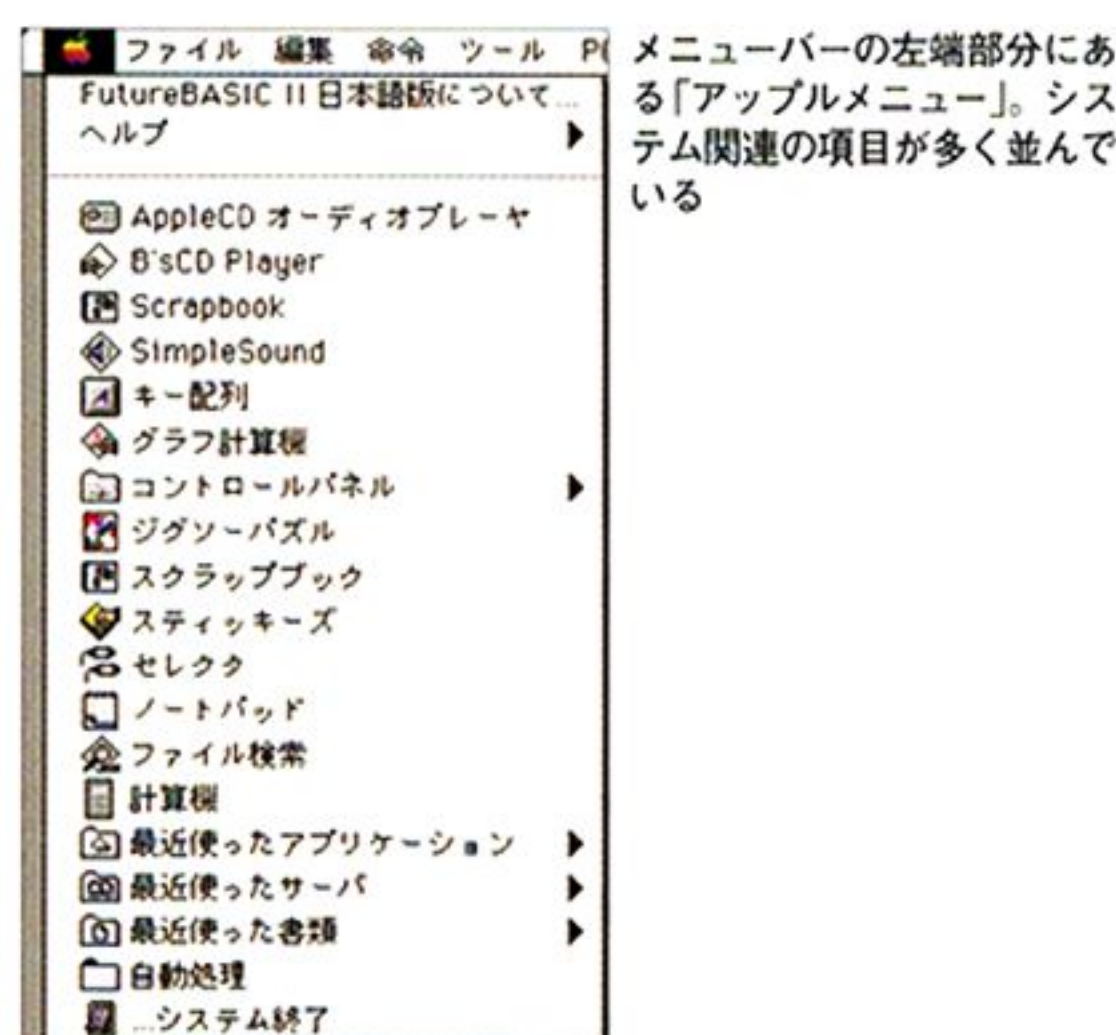
リスト1　ウィンドウを開く

```
WINDOW #1,"Sample",(30,60)-(330,230),_doc
while FN button = _false:wend
```

■メニュー/メニューバー

Macintoshは常に画面上部に「メニューバー」が表示されている。このメニューバーは使用しているアプリケーションが切り替わると選択できるメニューも切り替わるようになっている。つまり現在使用している(アクティブになっている)アプリケーションがメニューバーを占有することになる。

Macintoshシステムの画面の最上段部分に表示されるメニューバー。Macintoshのアプリケーションなどの機能はほとんどここから呼び出されることになる

このメニューバーだが，雑然とメニュー項目が並んでいるわけではない。このメニューの並び方にはお約束がある。まずいちばん左側は「アップルメニュー」と呼ばれる。林檎マークが表示されている部分だ。

メニューバーの左端部分にある「アップルメニュー」。システム関連の項目が多く並んでいる

次にファイルメニュー，編集メニュー……と続く。これに加えてMacOS 8.0以降では「ヘルプ」メニューが続くことになる。

右側からは，アプリケーション切り替えアイコン，スクリプトアイコン(各国語の入力を切り換える)……といった具合になる。右側は機能拡張書類などで，機能が追加されることもある。これらを含めるとMac Plusなど画面が狭い機種もしくはモニタではメニューバーの項目の文字数が長いと選択することができないメニューが出てくる。9インチモニタでも使えるのであればベストだが，少なくとも13インチモニタできちんと収まるようなメニュー項目名をつけたほうがよいだろう。MacOS 8.0からは日本語の書体が1ポイント小さくなったので，多少文字数には余裕がある。

メニューバー項目の順番だけでなく，そのメニューに表示され選択できる項目の順番も，おおよそ決まっている。まずアップルメニューの場合，1番目の項目は「このアプリケーションについて...」となっており，選択すると現在のアプリケーションについての説明などが表示される。作者の顔が表示されるものもあれば，妙なギャグをとばすものまで実に多様である。この部分だけ妙に凝っているものもある。作者の個性がもっとも反映される部分ともいえる。

アップルメニューの2番目は区切り線になり，区切り線以降はアップルメニューフォルダに入っている書類，アプリケーションなどが表示される。まれに2番目にヘルプなどを用意してあるアプリケーションもある(Future BASIC IIなどが，そうなっている)。

次のファイルメニューだが，上からおおよそ以下のようになっている。

「新規...」
「開く...」
「-----」(区切り線)
「閉じる」
「保存」
「別名で保存...」
「復帰」
「-----」(区切り線)
「用紙設定...」
「印刷...」
「-----」(区切り線)
「終了」

Macintoshのメニュー項目の順番だが，もっとも使用頻度の高い項目は上から2番目に置く。処理により分類し，大きく異なる処理の場合は区切り線で区別する。ファイルメニューのいちばん下の項目を選択すると必ず終了するようにする。おおよそ，このようになっている。

メニュー項目で，選択後すぐに実行されずユーザーが入力などを行う必要がある場合は，項目の末尾に「...」を付加する。これによりメニューを見た場合，即処理されるかどうかが判断できる。これは英語版でも日本語版でも中国語版でも，果てはアラビア語版でもまったく同様である。

次に編集メニューだが，これも項目の順番がほぼ決まっている。

「取り消し」
「-----」(区切り線)
「カット」
「コピー」
「ペースト」
「クリア」

さらに「全てを選択する」という項目が多くの場合追加される。必ずしも編集メニューが必要というわけではない。これは旧システム(7.0以前)で使用されていたデスクアクセサリという一種のミニアプリケーションとのデータをやりとりするために，ほぼ強制的に用意しなければいけなかったメニューだ。また，データをやりとりする必要のないゲームなどでは，このような編集メニューは存在しない。

メニュー項目には，項目名だけでなく，四葉マークに英数字といったような記号が表示されていることがある。これは，四葉マークのキー(コマンドキー)と表示されている英数字を同時に押すとメニュー項目を選択したことと同じになる。ショートカットと呼ばれている。このショートカットのキーも，基本的なものは，ほぼ決まっており以下のようになっている。

・ファイルメニュー
- 新規：N
- 開く：O（オー）
- 閉じる：W
- 保存：S
- 印刷：P
- 終了：Q

・編集メニュー
- 取り消し：Z
- カット：X
- コピー：C
- ペースト：V
- 全てを選択：A

・フォント（装飾）
- 太字：B
- 斜体：I

・特殊な組み合わせ
- 処理中断：.（ピリオド）

概略がわかったところで，次にメニューを構築するための命令について説明する。

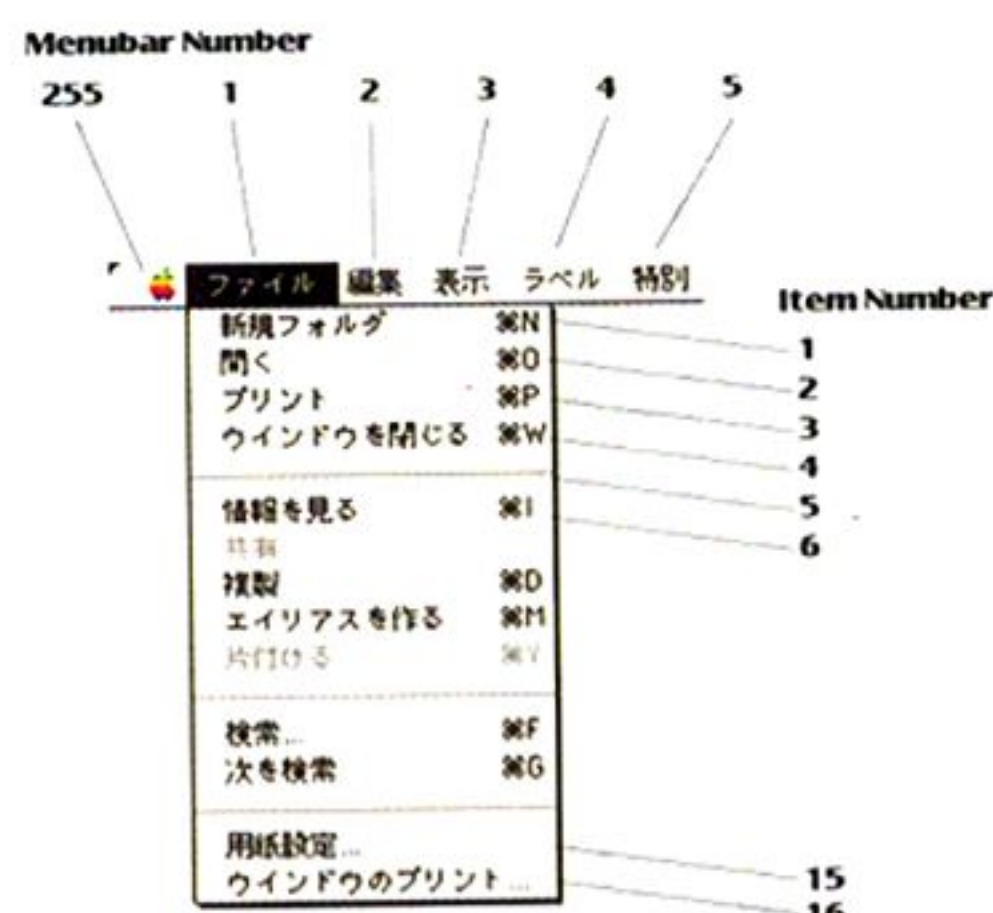
■メニューの構築

それではメニューを作ってみる。メニューを構築するにはMENU命令を使う。MENU命令のなかでアップルメニューと編集メニューは自動的に構築できるように専用の命令が用意されているので，これを利用するとよいだろう。

まずアップルメニューから作成する。アップルメニューのいちばん上は「このアプリについて...」という項目にする。この場合，

APPLE MENU “このアプリについて...”

となる。単純に項目名を書くだけだ。これだけで自動的にアップルメニューが構築される。

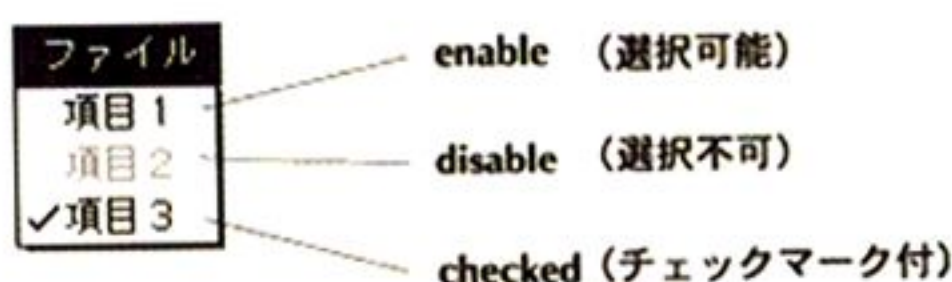
作成するファイルメニューのメニューバー，メニュー項目との番号の関係は以下の図のようになる。

メニューではメニューナンバー（横方向）とメニュー内で使用するアイテムナンバー（縦方向）の指定で処理を指定することになる

ファイルメニューなど汎用のメニューを作成する場合はMENU命令を使う。MENU命令の書式は以下のようになっている。

MENU メニューバー番号，項目番号，状態，項目名

メニューバー番号は左側から1，2，3……となる（最大20まで）。ファイルメニューの場合，（アップルメニューを除く）いちばん左側にあるので1番になる。項目番号はメニューバーに表示するメニューが0番，以後下に行くにしたがって1ずつ増加していく。

状態はメニューが選択できるかどうか，チェックマークをつけるかどうかなど，メニューの状態を示す。このメニューの状態は以下のものが設定できる。

_enable	選択可能
_disable	選択不可
_checked	チェックマークを付加

メニュー項目の状態には通常の「選択可能」と「選択不可」な場合，そしてすでに選択されていることを示すチェックマーク付きの3種がある

今回のファイルメニューは，終了項目しかなく必ず選択できなければならないので以下のようになる。

MENU 1,0,_enable,"ファイル"

MENU 1,1,_enable,"終了"

メニュー項目は存在する数だけ設定しなければならない。

次に編集メニューだが今回は使用しないので使わない。ちなみに「EDIT MENU メニューバー番号」とする。

最後にキーボードショートカットだが，ショートカットを定義するには定義文字の先頭に/（スラッシュ）を書き続いてショートカットとなる文字を記述する。Qで終了するのであれば

MENU 1,1,_enable,"/Q終了"

となる。AであればMENU 1,1,_enable,"/A終了"となる。キーボードショートカット以外にメニュー表示文字を太字にしたり区切り線を入れることもできる。表1のものが使用できる。複数の効果を実現したい場合は列記すればOKだ。

表1

(	メニュー項目を選択不可状態にする
/	次に続く文字をショートカットとする
!	チェックマークを付加する
^	アイコンを付加する
;	区切り線
-	区切り線
<B	太字
<I	斜体
<O	袋文字
<U	下線
<S	影付き文字

以下のサンプルはメニューを構築する。選択はできるが選択後の処理は行っていない。また，メニューを選択すると選択した項目のメニューバー文字が反転したままになる。コマンドキーと.（ピリオド）を同時に押すと終了する。

リスト2　メニューを構築する

```
APPLE MENU "このアプリについて..."
MENU 1,0,_enable,"ファイル"
MENU 1,1,_enable,"/Q終了"

WHILE _true
  HANDLEEVENTS
WEND
```

■メニューの選択処理

メニューが構築できたら次は選択されたメニュー項目に応じて処理を振り分けなければならない。「どのメニューバー項目が選択されたか？」「そのなかの，どの項目が選択されたか？」という処理を行う。まず，どのメニューバー項目が選択されたかを調べるには以下のようにする。

menuID = MENU(_menuID)

これで変数menuIDに選択されたメニューバーの番号が入る。この番号はメニュー構築時に割り振った番号と一致する。ただし，アップルメニューは0番ではなく255番となる。MENU(_menuID)命令のカッコ内の_menuIDは「どのメニューバー項目が選択されたか」を返すために与えるパラメータだ。

次に選択されたメニュー項目を調べる。以下のようにすることで選択された項目番号を調べることができる。MENU(_itemID)の_itemIDは「どの項目番号が選択されたか」を返すために与えるパラメータだ。

itemID = MENU(_itemID)

これで選択されたメニューバー項目とメニュー項目が取得できる。あとは条件が一致するかどうか調べて処理を振り分ける。条件により処理を振り分けるにはIF命令を使う。この命令の書式は以下のようになる。

IF 条件式 THEN 命令

IF命令は条件式が成立したときにTHEN以降に続く命令を実行する。たとえば変数aが123だったときに音を出す場合は，

IF a = 123 THEN BEEP

となる。条件式はひとつだけでなく複数記述することができる。このときに複数の条件を指定することができる。この場合，

「AND」（双方のどちらも条件が成立）

「OR」（双方のどちらかの条件が成立）

が使用できる。aが123でbが456の場合，aが

123またはbが456の場合に音を出すプログラムはそれぞれ以下のようになる。

・aが123でbが456の場合

IF (a = 123) AND (b = 456) THEN BEEP

・aが123またはbが456の場合

IF (a = 123) OR (b = 456) THEN BEEP

あとは同様にメニューバー番号，選択項目番号を調べてやればよい。今回の時計では「アップルメニュー」「このアプリについて...」と「ファイル」「終了」の2つだけだ。これらは以下のようになる。

IF (menuID = 255) AND (itemID = 1) THEN FN printAbout

IF (menuID = 1) AND (itemID = 1) THEN END

最初の行はアバウト画面を表示する部分でTHEN以降のFN printAboutで関数printAboutを呼び出している。アバウト画面表示部分については後ほど説明する。ファイルメニューの終了が選択された場合は，プログラムを終了する。プログラムを終了させる場合はEND命令を使う。*1

これでメニューの選択処理は終わり，といきたいところだがメニュー選択後，選択したメニューバー項目が反転表示されたままになってしまう(下図)。

メニューの処理は終了したのだが，「ファイル」の部分が黒いままで残ってしまった

反転した状態を戻すには選択された処理が終了したあとに，

MENU

とする。これでメニューの反転した状態が元に戻る。

これでメニューの構築もでき，選択された項目を知ることができるようになった。しかし，このままではFuture BASICはメニューの選択処理を呼び出してくれない。プログラムは存在するが，ただのゴミということだ。Future BASICには，メニューが選択されたときに選択処理プログラムを呼び出すための命令が用意されている。ON MENU命令だ。これは以下のようになっている。

ON MENU FN 関数名

関数名の部分にメニュー選択処理を行う関数名を書いておけば自動的にメニューが選択された場合，呼び出されるようになる。

＊4　今回のようなメモリを確保したりしないプログラムでは問題ないが，通常のプログラムではこのような書き方は推奨できない。quitFlag = _true などとして，終了が選択されたことを示すフラグを用意してイベントループを脱出するようにしたほうがよい。

以下のサンプルは選択されたメニュー項目番号を表示する。コマンドキーと.(ピリオド)を同時に押すと終了する。

リスト3　選択されたメニュー番号を表示する

```
LOCAL FN initMenu
  APPLE MENU "サンプルついて..."
  MENU 1,0,_enable,"ファイル"
  MENU 1,1,_enable,"項目1"
  MENU 1,2,_enable,"項目2"
  MENU 1,3,_enable,"項目3"
  MENU 1,4,_enable,"項目4"

  MENU 2,0,_enable,"編集"
  MENU 2,1,_enable,"項目1"
  MENU 2,2,_enable,"項目2"
  MENU 2,3,_enable,"項目3"
  MENU 2,4,_enable,"項目4"
END FN

' "メニューが選択された場合の処理"
LOCAL FN selectMenu
  menuID = MENU(_menuID):      ' "選択されたメニューバー項目を選択する"
  itemID = MENU(_itemID):      ' "選択されたメニュー項目番号を取得する"
  PRINT "menubar No. = ";menuID
  PRINT "item No. = ";itemID
  MENU
END FN

FN initMenu
ON MENU FN selectMenu

WHILE _true
  HANDLEEVENTS
WEND
```

■文字を表示する

次に時刻を表示する。ウィンドウに文字を表示する場合はPRINT命令を使う。PRINT命令は文字だけでなく数値なども表示できる。文字を表示する場合はダブルクオーテーションで囲む。

・文字を表示する

PRINT "Sample..."

・数値を表示する

PRINT -12.5

PRINT命令は文字や数値を表示したあと，自動的に改行してしまう。このとき，文字や数値がウィンドウ内に表示しきれなくなるとすでに表示してある内容を上に移動(スクロール)させる。自動的に改行させないようにするにはPRINT命令の最後に;(セミコロン)を付加する。

PRINT "Sample...";

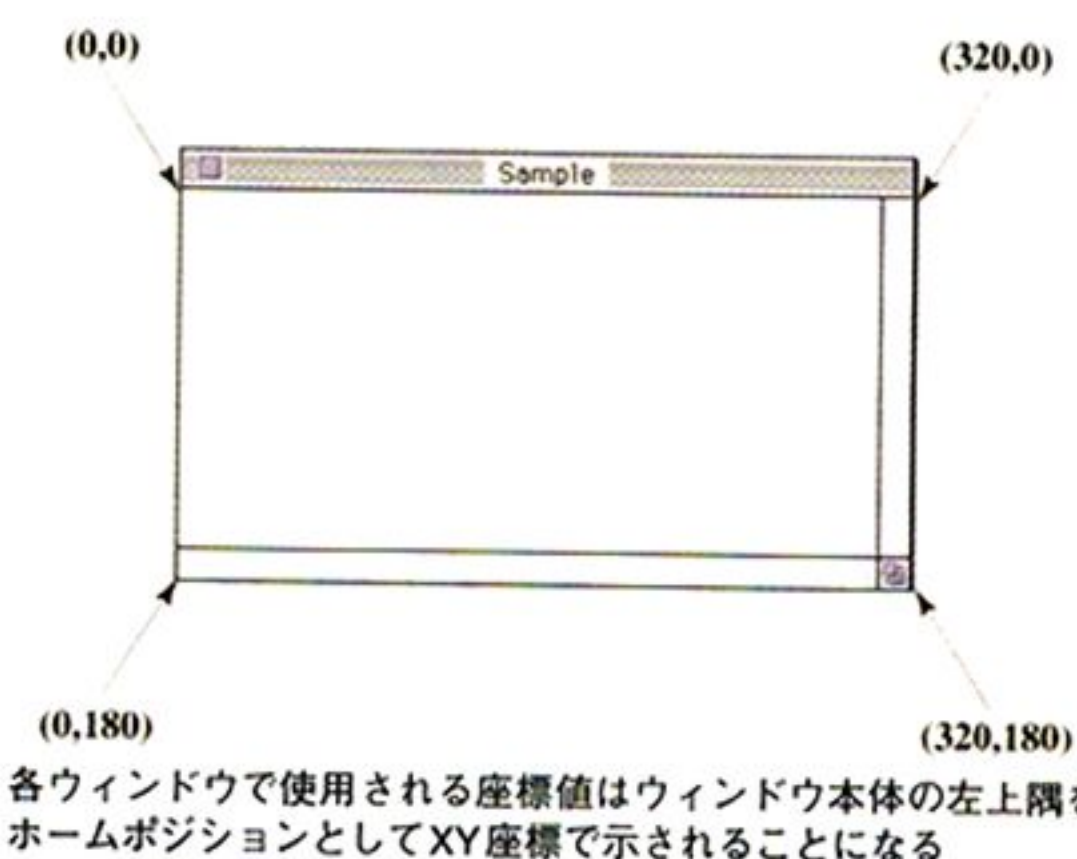
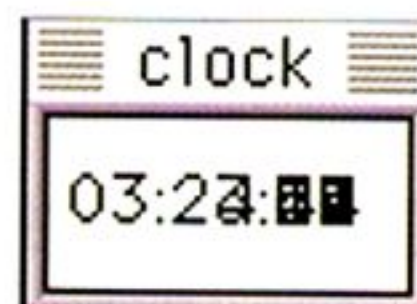
PRINT -12.5;

;を付加した場合，ウィンドウの右下まで文字が表示されるまでは改行しない。

またPRINT命令はウィンドウに表示する位置も指定できる。表示位置を指定する場合は

PRINT %(X座標,Y座標),文字列

とする。指定する座標だが，これは下図のようにウィンドウの左上が(0，0)となっており，右下に行くにしたがって座標値が増加する。

各ウィンドウで使用される座標値はウィンドウ本体の左上隅をホームポジションとしてXY座標で示されることになる

これで文字や数値は表示できる。肝心の時刻を表示するにはTIME$と組み合わせる。TIME$には現在の時刻が入っており，PRINT TIME$とすれば時刻を表示することができる。ところが座標を指定して表示させてみると文字が重なって読めなくなってしまう。

ちゃんと文字の表示位置を指定したのだが，前の文字と重なって表示されてしまった

これは「表示モード」が「重ね合わせ」(_srcOr)に設定されているためだ。文字の表示モードは以下のものが使用できる。

文字の表示モードには上図のような種類がある。すでに描画されている部分とそうでない部分での動作の違いを確認しておこう

表示モードを重ね合わせ(_srcOr)ではなく「コピーモード」(_srcCopy)にすれば文字が重なって読めなくなることはない。テキストの表示モードを指定するにはTEXT命令を使う。

TEXT ,,,表示モード

として指定する。表示モードをいろいろ変えてみるとわかりやすいかもしれない。

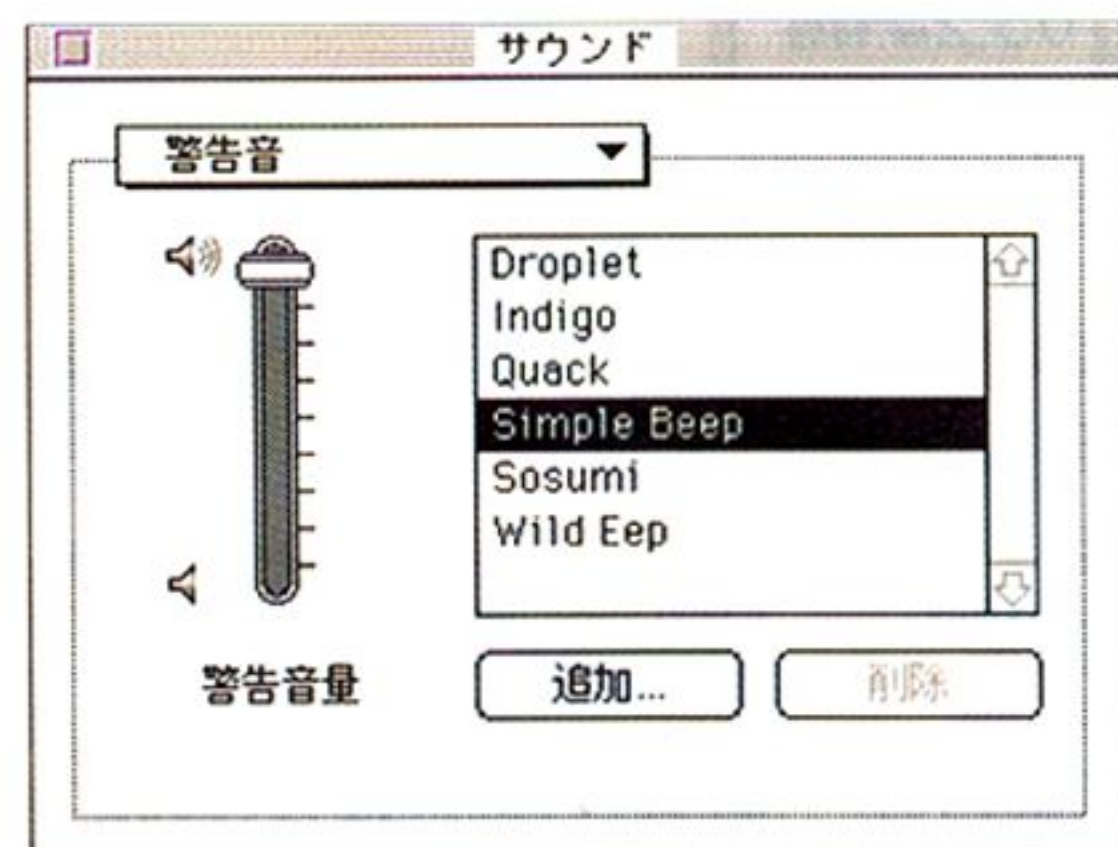
以下のサンプルはさまざまな描画モードで文字を表示する。コマンド+.(ピリオド)を同時に押すと終了する。

■アバウト画面

アバウト画面はアプリケーション起動時の画面(スプラッシュウィンドウ)と同じものであること

リスト4　さまざまな描画モードで文字を表示

```
WINDOW 1,"Sample",(0,0)-(300,450)
BOX FILL 120,0 TO 300,450
TEXT ,24
PRINT
PRINT
TEXT ,,,_srcCopy
PRINT "Future BASICでGO!"
TEXT ,,,_srcOr
PRINT "Future BASICでGO!"
TEXT ,,,_srcXor
PRINT "Future BASICでGO!"
TEXT ,,,_srcBic
PRINT "Future BASICでGO!"

TEXT ,,,_notSrcCopy
PRINT "Future BASICでGO!"
TEXT ,,,_notSrcOr
PRINT "Future BASICでGO!"
TEXT ,,,_notSrcXor
PRINT "Future BASICでGO!"
TEXT ,,,_notSrcBic
PRINT "Future BASICでGO!"

TEXT ,,,_grayishTextOr
PRINT "Future BASICでGO!"

BEEP
DO
  HANDLEEVENTS
UNTIL _false
```

が多い。このアバウト画面は作者の顔の部分であるから，自由に表現して構わない。BGMで演奏が始まってもよいし，スタッフロールでもよい。単純に文字を表示するだけのシンプルなものでもよい。今回は単純に文字を表示し，音を鳴らしている。音を鳴らすにはBEEPと指定する。ビープ音と呼ばれるのだが，実際はコントロールパネルで指定した警告音が演奏される。

ここで設定したサウンドが演奏される

アバウト画面を表示したらマウスのボタンが押されるまで待つことにする。マウスのボタンが押されたか調べる場合はFN BUTTONを使う。この関数の定義はプログラム中には出てこない。これはFuture BASICにあらかじめ用意されているものだ。実際にはFuture BASICが用意しているというよりMacintoshのToolbox（各種関数群*5）を呼び出せるようにあらかじめ定義されているといったほうが正しい。Future BASICでは自分が作成した関数も，あらかじめMacintosh側で用意してある関数も区別なく利用できるようになっている。*6

FN BUTTONは，マウスボタンが押されていれば_true，押されていなければ_falseを返すようになっている。あとはボタンが押されるまで待つ処理を加えればよい。一定の条件を満たすまで繰り返し処理を行う命令として以下のものがある。

[1] 条件が成立している間繰り返す

```
WHILE 条件式
  :
WEND
```

[2] 条件が成立するまで繰り返す

```
DO
  :
UNTIL 条件式
```

[1]のWHILE～WENDは条件が成立している間繰り返すのでマウスボタンが押されるまで待つ場合は以下のようになる。

```
WHILE FN BUTTON = _false
WEND
```

[2]のDO～UNTILの場合は条件が成立するまで，つまりマウスボタンが押されるまで待つため以下のようになる。

```
DO
UNTIL FN BUTTON = _true
```

どちらもマウスボタンが押されるまで待ち続ける。今回のような単純な処理ではWHILE～WEND，DO～UNTILどちらも同じように見えるが，WHILE～WENDは最初に条件がチェックされるためWHILE～WENDの間の命令は一度も実行されない場合がある。これに対してDO～UNTILは最後に条件をチェックするため最低でも1回はDO～UNTIL間の命令が実行されることになる。

話を元に戻そう。マウスボタンが押されたらアバウトウィンドウを閉じる必要がある。ウィンドウを閉じる場合はWINDOW CLOSE命令を使う。このときにどのウィンドウを閉じるのかを番号で指定しなければならない。この番号はWINDOW命令で新しくウィンドウを作成した場合の番号と同じである。WINDOW #2,……であればWINDOW CLOSE #2で2番のウィンドウが閉じられる。

また，プログラムが終了した場合，開かれているウィンドウはすべて閉じられる。時計のウィンドウはWINDOW CLOSE命令で閉じられていないが，プログラム終了と同時に自動的に閉じられるため問題ない。

＊5　WindowsではAPI（Application Program Interface）と呼ばれる。どちらもOS呼び出しである。

＊6　残念ながら通常の状態（初期状態）ではMacintoshのすべての関数を利用することができない。最新の関数やサポートされていない関数を利用した場合は，付属のPascal Converterを利用して変換する必要がある。

■イベントについて

Macintoshはイベント駆動型といわれる。これは，プログラマが用意した順番に処理を行うのではなく，ユーザーの要求に応じてプログラムを実行する，といったものだ。つまりユーザー主導型ともいえる。ユーザーが要求したものをMacintoshでは「イベント」として受け取る。このイベントに応じて各種処理を振り分けなければならない。

Future BASICが受け取ることができるイベントは以下のとおりだ（左側はFuture BASICで定義してある定数名）。

_nullEvt	ヌルイベント
_mButDwnEvt	マウスダウンイベント
_mButUpEvt	マウスアップイベント
_keyDwnEvt	キーダウンイベント
_keyUpEvt	キーアップイベント
_autoKeyEvt	オートキーイベント（リピート）
_updateEvt	アップデートイベント（画面更新）
_diskInsertEvt	ディスク挿入イベント
_activateEvt	アクティベートイベント
_networkEvt	ネットワークイベント（System 7.0 定義）
_ioDrvrEvt	I/O ドライバイベント（予約済み）
_applEvt	アプリケーション定義
_appl2Evt	アプリケーション定義
_appl3Evt	アプリケーション定義
_appl4Evt	OSイベント（System 7.0 定義）

マウスのボタンが押されたら，どのウィンドウのどこで押されたのか，どのボタンが押されたのか，クローズボックスが押されたのかどうかなどを分けて処理しなければならない。しかし，これではプログラムを作成するのが大変である。このイベント処理のほとんどは「定型処理」に近いものがある。たとえばウィンドウを移動させる，ウィンドウサイズを変更する，画面を更新するといったものである。この部分をまとめてひとつの命令にしておけばプログラムは短くなり簡単になる。Future BASICでは，このイベント処理部分を「HANDLEEVENTS」という1命令でこなしている。プログラムが終了するまでイベント処理を行っていればFuture BASICが自動的に処理をしてくれるため，実際のプログラムはわずか3行で終わる。

```
WHILE _true
  HANDLEEVENTS
WEND
```

定型処理だからといって，このような1命令でまとめてしまうと応用が利かないと思われるかもしれないが，Future BASICではイベントに応じて処理を設定する命令が用意されており，自前で処理することが手軽にできるようになっている。自分が行いたい部分だけ作成すればよいのだ。これがC言語では1から用意しなければいけないためプログラムは複雑になり見通しも悪くなってしまう。

Future BASICで，あらかじめ定義されているイベントの振り分けが可能なものは以下のとおりだ。

- コマンド+.（ピリオド）が押された[ON BREAK FN]
- ダイアログイベント（ウィンドウに関するイベント）[ON DIALOG FN]
- エディット（フィールド）イベント[ON EDIT FN]
- LPRINTイベント[ON LPRINT FN]
- メニューイベント[ON MENU FN]
- マウスイベント[ON MOUSE FN]
- タイマーイベント [ON TIMER FN]

メニュー，ダイアログイベントがもっとも多く使用される。このイベント部分は，説明するとかなり長くなるので今回はこれ以上の説明はしない。

作成する時計の場合，メニューイベントだけの処理を行うことになる。選択後の処理を設定する

にはON MENU FNを使う。これはメニューの選択の項で説明した。要するにメニューイベントを処理している。

■タイマ

時刻は1秒ごとに表示，更新させなければならない。イベントループ内で毎回，時刻が更新されたか調べるという方法もあるが，幸いFuture BASICには指定秒数ごとに動作させるためのタイマが用意されている。タイマを起動させる命令は，

ON TIMER(秒数) FN 関数名

となる。時刻は1秒ごとなので秒数には1を，そして関数名は時刻を表示する関数名printTimeを指定する。これで自動的にタイマが起動し指定された関数の処理を実行してくれる。ON TIMERは秒数だけでなく1/60秒を基準（1とする）として設定することもできる。この場合は数値を負数にすればよい。たとえば0.5秒ごとにタイマを起動させるのであれば－30を指定する。つまり30/60=0.5秒ということだ。

■エラーについて

プログラムを作成していざ実行しようとすると，コンパイルエラーになってしまう場合がある。Future BASICで犯しやすいエラーを載せておくので参考にしてほしい。

●関数名が違う

特に大文字と小文字の違いでコンパイルエラーになってしまう。簡単なミスであり関数名を直せばよい。

●関数が定義されていない

関数が定義，作成されていない，対応していないToolbox呼び出しを行ったなどが考えられる。これ以外に，呼び出す前に，呼び出される関数が先頭に定義されているかどうかもチェックしたほうがよい。

●変数と定数を間違えた

この場合はエラーにならずにプログラムが誤動作する。プログラムの動作が期待どおりでない場合，再度プログラムを見直したほうがよい。

●関数の戻り値と受け取る変数の型が違う

関数の戻り値が整数型なのに，受け取る変数（代入先の変数）が文字型ではエラーになる。双方の型を一致させればよい。

●変数が定義できない

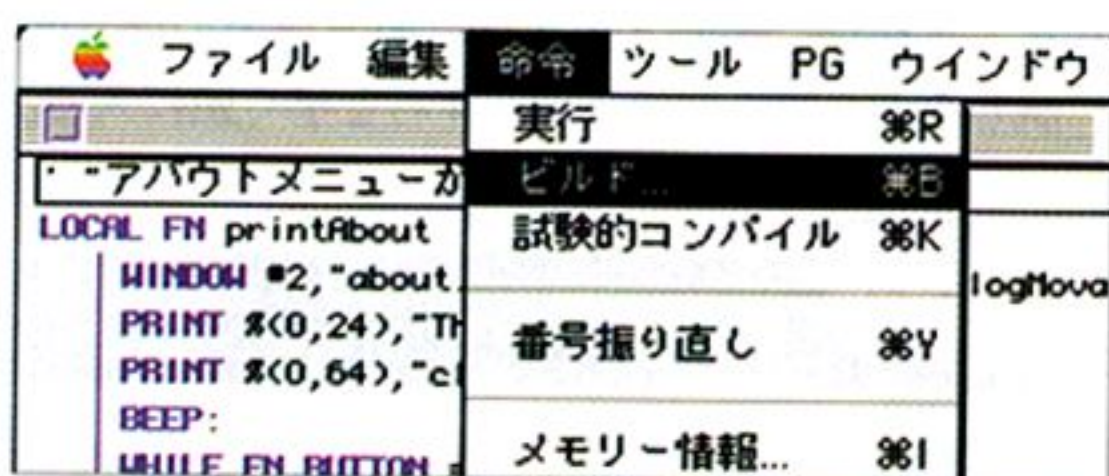
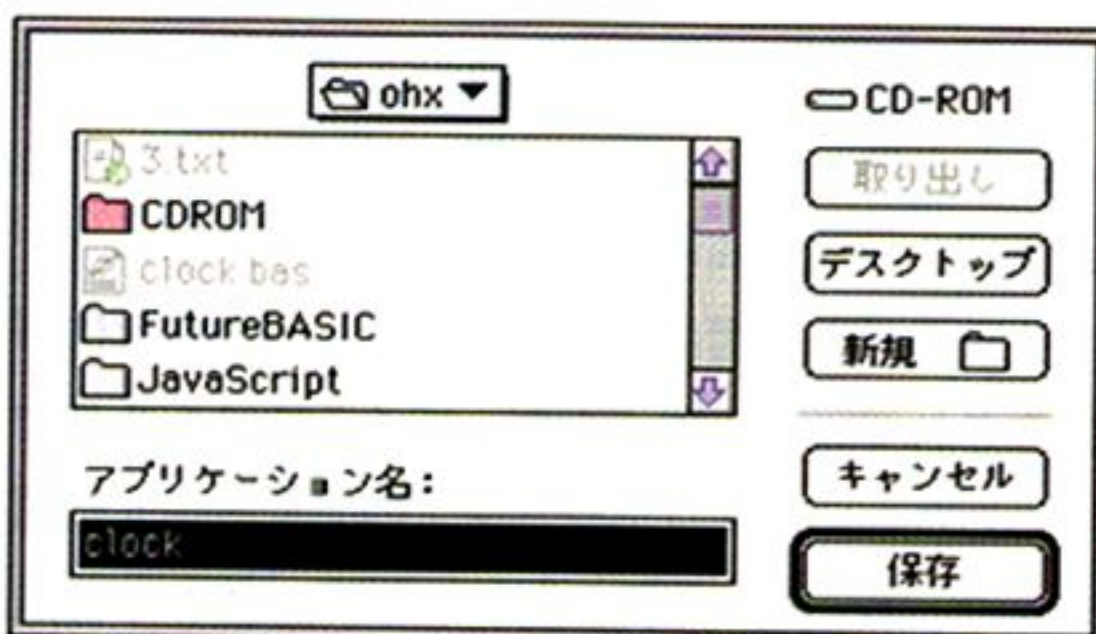
Future BASICでは変数とプログラムを合わせて32Kバイト以内という制限がある。この場合はSEGMENT命令で変数領域とプログラム領域をいくつかに分割しなければならない。大量にメモリを確保，使用する場合はToolboxを利用するしかない。

プログラム作成に慣れないうちは，なるべく短いプログラムを入力して覚えたほうがよい。最初から長いプログラムを作ろうとすると大変だ。幸いFuture BASICのマニュアルには有益で短いサンプルが大量に掲載されている。このサンプルをもとに改良してみるとよいだろう。

リスト5　時計表示プログラム

```
  ' *アバウトメニューが選択された場合の処理*
LOCAL FN printAbout
  WINDOW #2,"about...",(0,0)-(256,120),_dialogMovable:' *ウィンドウを開く*
  PRINT %(0,24),"The Clock (^-^)/":               ' *メッセージを表示する*
  PRINT %(0,64),"click to mouse button...":       ' *メッセージを表示する*
  BEEP:                                           ' *ビープ音を出す*
  WHILE FN BUTTON = _false:                       ' *マウスボタンが押されるまで待つ*
  WEND
  WINDOW CLOSE #2:                                ' *アバウトウィンドウを閉じる*
END FN

' *メニュー項目の初期化*
LOCAL FN initMenu
  APPLE MENU "このアプリについて...":             ' *アップルメニューを作成*
  MENU 1,0,_enable,"ファイル":                    ' *ファイルメニューを作成*
  MENU 1,1,_enable,"/Q終了":                      ' *終了メニュー項目を作成*
END FN

' *メニューが選択された場合の処理*
LOCAL FN selectMenu
  menuID = MENU(_menuID):                         ' *選択されたメニューバー項目を選択する*
  itemID = MENU(_itemID):                         ' *選択されたメニュー項目番号を取得する*
  IF (menuID = 255) AND (itemID = 1) THEN FN printAbout:' *アバウトが選択された*
  IF (menuID = 1) AND (itemID = 1) THEN END:      ' *終了メニューが選択された*
  MENU
END FN

' *時計を表示する関数*
LOCAL FN printTime
  TEXT ,,,_srcCopy
  PRINT %(0,16)TIME$;:                            ' *時刻を表示する*
END FN

' *各種初期設定*
ON TIMER(1) FN printTime:                         ' *1秒ごとに関数printTimeを呼び出す*
ON MENU FN selectMenu:                            ' *メニューが選択されたときの処理関数selectMenuを呼び出す*
FN initMenu
WINDOW OFF:                                       ' *起動時に表示されるウィンドウを出さないようにする*
WINDOW #1,"clock",(50,50)-(110,75),_dialogMovable:
                                                  ' *時計を表示するウィンドウを開く*

' *イベントループ*
WHILE _true
  HANDLEEVENTS
WEND
```

■アプリケーション化するには？

作成したプログラムをアプリケーションにするのは簡単だ。命令メニューから「ビルド...」を選択し作成するアプリケーションの名前を入力するだけだ。

メニューバーの「命令」メニューから「ビルド」を選択すると単体の実行プログラムが作成できる

ここでファイル名を指定する。この場合はclock.basだ

これだけで作成したプログラムがMacintosh上で動くアプリケーションになる。ほとんどの場合，C言語で作成したといってもばれないほどコンパクトで高速に動作する。

作成したアプリケーションには，オリジナルのアイコン[*7]をつけたくなるが，とりあえずグラフィックソフトで描いておき，グラフィックデータを「コピー」する。次にアプリケーションを選択し，「ファイルメニュー」「情報を見る...」を選択する。左上のアイコン部分をクリック後，「編集メニュー」「ペースト」を選択する。これで，無事にアプリケーションができあがった。

＊7　ここで書いたのは，もっとも手軽で安全かつ確実な方法だ。通常はこのようなことはせずに，BNDLリソースとFREFリソース，およびアイコンを作成，編集しリンクする。

■終わりに

できあがったプログラム（リスト5）は説明の割には短い。短いプログラムだが，いろいろと考慮しなければならない部分が多いのがMacintoshのプログラミングだ。特にプログラムとは別の部分，ユーザーインタフェイスのお決まり部分（ガイドライン）がいくつかある。基本的にはガイドラインに沿うべきだが，アプリ内でインタフェイスが統一されていればいい（Lightwave 3D, Bryce 3Dなど好みは分かれるところだがひとつの目安ともいえる）。慣れないうちはガイドラインに沿って作成するとよいだろう。

コツさえわかればMacintoshでプログラムを作成するのは決して難しいことではない。ただ，一気に上を目指そうとすると挫折してしまうかもしれない。確実にステップアップしていくほうがよいだろう。結局のところ，最後は本人のやる気次第。駄目だ，駄目だと思っていれば駄目なのだ。いまは駄目でも作れる，作ろうと思うことがいちばん大事ではないだろうか。

Macでプログラミングを始める10のヒント

柴田　淳/Shibata Atsushi

Macintoshでちゃんとしたプログラミングをする場合には，やはりC/C++が主流となっている。ここではMacintoshプログラム初心者がCode Warriorを使用してプログラムを作る際のヒントを挙げてみよう。

パソコンが高機能になっていくにつれて，プログラミングがユーザーから縁遠い存在になっていくのは至極当然の話である。ひと昔前なら，プログラムといえば文字を表示したり，簡単な計算をするだけのものであったが，いまやパソコンは何万色ものグラフィックを扱えるし，音も出せ，そしてネットワークにも繋がる。機能が増えれば機能を操る命令も当然増える。つまり，10年ほど前のプログラマよりも，現在のプログラマのほうが覚えなければならないことがたくさんあるのだ。

たとえば，Macintoshでウィンドウを開くプログラムを書くだけで，10数行のコードが必要になる。しかも，このコードは単にウィンドウを開くだけで，ドラッグしたり，ウィンドウの中に文字を表示したりといった処理を一切しない。ごく小さなプログラムでさえ当然のように行っていることをするには，さらに何倍ものコードを書かなければならないのである。

MacintoshのOSに組み込まれているAPI関数(いろいろな機能を操る命令群)は，数万に及ぶといわれている。数は膨大なのだが，全部覚えないとプログラミングができないかというとそうではない。プロのプログラマだって，ソラで覚えているのはほんの一部にすぎない。あとは，漠然と，命令の概要だけを記憶していて，使う段になって調べるわけである。必要な処理を実現する方法をいかに早く見つけ出すかが，プログラマの能力に深く関わっているといえるかもしれない。

知識しかないプログラマは，一定の問題に対する答えは出せても，応用力に欠けている場合が多い。しかし，生産的なプログラミングにおいて一般的に求められるのは，イレギュラーな問題に対処したり，新しい分野に対する解法をひねり出すことなのである。未知の問題に対処するコツのようなものを知っているプログラマというのは，有能なプログラマといえるのかもしれない。

プログラミングの初心者は，知識が限りなくゼロに近いプログラマといえるだろう。もっというと，知識が少ないプログラマでも，コツさえつかめば，プログラムを組んでしまえるものなのである。

これから，Macintoshでプログラミングを始めるにあたって必要な，10のヒントを提供したいと思う。このヒントが，あなたがプログラミングの世界へと踏み出すステップとなれば幸いである。

ヒント1　開発に必要な環境を揃える

資本主義の世の中だ。多少の出費は覚悟してもらわなければならない。

どんなプログラムを組みたいかによって，揃える環境もさまざまである。とりあえずこの記事では，C++言語を使ったプログラム開発に焦点を絞ることにする。また，原稿執筆時点で最新のCW Pro3をもとに話を進めることにする。

Macintosh用のC++開発環境には，Metrowerksという会社から出ているCodeWarriorというパッケージが必要である。

次にハードウェア環境だが，メモリは最低64Mバイトはほしいところ。マシンの速度は，速いほうがよい。コンパイルという作業は，意外に時間がかかるものなのである。

パッケージを買ってきたら，チュートリアルなどを読んで，ひととおり環境に慣れること。詳しくなる必要はない。わからないことがあったら，またマニュアルを開けばいいのである。また，マニュアル類は，熟読するより，試せる事柄は実際に試しながら読んだほうが，ずっと早く頭に入る。

ヒント2　言語を覚える

英語のネイティブスピーカーのオネーチャン(あるいはオニーチャン)を好きになったら，死にもの狂いで英語を覚えるだろう？　それと，同じレベルの話。これからプログラミングを始めようというのなら，使用する言語の文法くらいは覚えるのが当然。これからの世界，プログラミングといえばC++くらいは覚えたいもの。

入門書はよく選ぼう。書店に行って，闇雲に平積みされている入門書を買ってくるのはあまりすすめられない。プログラミングのできる知人に聞くなり，Webで検索するなりして，良書と目星をつけたものを購入しよう。

これも，詳しくなる必要はない。大雑把な言語仕様を覚え，コードを読んで，宇宙語に見えない程度で十分。コードを読んでいてわからないことがあったら，本に戻って調べればいい。そういう作業を繰り返すうちに，大切なことは自然と身につくものなのだ。

金銭的に余裕があったら，入門書と，仕様を網羅したリファレンス的なものの2種類を手元に置くのが理想的。

ヒント3　まず改造から始める

CodeWarriorには，たくさんのサンプルコードがついてくる。CodeWarriorの使い方をひととおり覚えたら，このサンプルコードを片っ端からコンパイルしてみよう。どのサンプルも，工夫が凝らしてあってなかなか楽しいものばかりだ。

コードの入ったフォルダの中に，rsrcという文字で終わるファイルが見つかるはずだ。これはリソースファイルといって，プログラムで利用されるウィンドウ，音，文字などの情報を保存しておくファイルである。

このファイルをダブルクリックすると，ResEditというアプリケーションが立ち上がるはずである(画面1)。まずは，このファイルを適当にいじってみよう。たとえば，DLOGという名前のリソースを開くと，プログラムで利用しているダイアログが表示されるはずである。ボタンの配置など

画面1　ResEditの画面
リソースファイルにはウィンドウの大きさの情報や文字列などが入っている

画面2：Constructorの画面
PowerPlantで利用する情報を編集するConstructorの画面

を変えて，プログラムを再コンパイルしてみると，変更が反映されているはずである。

また，PPOBという文字で終わるファイルが含まれていることがあるはずだ。これも，見つけたらおもむろにダブルクリック。今度は，Constructorというプログラムが立ち上がるはずである（画面2）。このプログラムは，のちほど説明するPowerPlantというクラスライブラリで利用するウィンドウ，メニュー，アイコンなどの情報をひとまとめにしたファイルである。リソースファイルとは少々趣が違っていて，より細かい設定ができるようになっている。

このファイルも，適当に書き換えて再コンパイルしてみよう。編集結果が，ちゃんと反映されているのがわかると思う。プログラム本体を書き換えなくても，かなり自由に見栄えを変えることができるのである。お手軽で，なかなか楽しい遊びといえよう。

リソースファイルもPPOBファイルも，相当大規模な変更を加えない限り，プログラムの実行に影響を与えない。ただし，書き換えた個所が悪くて，プログラムが走らなくなる場合がある。そのような場合に備えて，変更を加える前のファイルのコピーを取っておこう。万が一不具合が生じたら，コピーしたファイルを復帰させれば，すべて元どおりになる。

ヒント4　PowerPlantを使う

CodeWarriorでは，プログラムで利用するソースコードや各種のファイルを，プロジェクトファイルという形式で管理している。試しに，新規プロジェクトを作ってみよう。作成するプロジェクトのタイプを選ぶダイアログが現れるはずである（画面3）。このなかから，〝BasicPowerPlant PPC（あるいは68K）〟という項目を選んで，OKを押してみる。

プロジェクトウィンドウには，あらかじめ2つのソースファイル，リソースファイル，PPOBファイルが含まれている。このプロジェクトは，この状態ですでにアプリケーションの体裁をなしており，きちんとコンパイルできるのである。

できあがったアプリケーションを走らせると，Hello Worldと書かれたウィンドウが現れる。コマンド-Nを押すと，新規ウィンドウが作られる。ソースを見ると，たった数百行ほどで，冒頭に示した単純なソースの高々数倍の長さである。なのに，このプログラムのなんと華麗なこと！　ウィンドウは，ドラッグなど，ひととおりの操作ができる状態にある。メニューも選べるし，ちょっとしたアバウトダイアログまで出る始末。

数百行の少ないコードで，これだけの自由度を実現しているのが，PowerPlantというクラスライブラリである。実際は，背後にウィンドウやメニューの操作を肩代わりしてくれるコードが潜んでいるのだ。

ウィンドウをリサイズするとか，基本的な操作を行うコードは，誰が書いても似たようなものになりがちである。ならば，いっそ共通化してしまって，プログラマはカスタマイズの必要がある部分だけコードを書けばいい。この共通部分を，C++のクラスというかたちで提供してくれているのがPowerPlantというわけである。

クラスライブラリの利点は，プログラマが必ずしもOSの仕組み全体を知らなくても，プログラムを書くことができる，という部分である。C++の基本的な文法を知っていれば，扱うのはそれほど難しくない。

画面3：新規プロジェクト
新規プロジェクトを作成するときに出るダイアログ。プロジェクトにもたくさんの種類がある

画面4：ブレークポイントで止まったデバッガのウィンドウ。デバッガで追えば，プログラムの流れを簡単につかむことができる

ヒント5　ソースレベルデバッガを活用する

さて，もう一度，ソースコード（CPPStarterApp.cp）に目を移してみよう。104行目に，こんな部分が見つかるはずである。

```
theWindow＝LWindow::CreateNewWindow
        (window_Sample, this)；
```

CreateNewWindowとあるからには，ここでウィンドウを作っているに違いない。また，関数名がObeyCommand()となっている。コマンドに従う，ということは，メニュー選択後の処理だろうか。

試しに，ここにブレークポイントを置いて，プログラムを走らせてみよう。まず，CodeWarriorのプロジェクトメニューから，デバッガを有効にする。次に，該当行の黒い横線をマウスでクリックし，赤い丸をつける。コマンド-Rでプログラムをコンパイル，実行する。

まず，プログラムが走り始めた直後に，この行が実行されていることがわかる。StartUp()という関数から，ObeyCommand()が直接呼ばれているのだ。このように，デバッガを使うと，プログラムの動きが手に取るようにわかるのである。最初にウィンドウが現れる仕組みが，これで理解できただろう。

次にいよいよ，メニューからNewを選択（画面4）。今度は，止まったObeyCommand()の上にLCommander::ObeyCommand()などとあるが，あまり気にしなくてよい。とにかく，ObeyCommand()という関数は，コマンドの要求を受け，受け取ったコマンド番号に適した処理をすればよいのだ，ということがわかっていただけただろうか。

ちなみに，107行目から次のようなコードを足してみよう。

```
case cmd_Paste :
    SysBeep(30)；
    break;
```

SysBeep()というのはMacのAPI関数で，システムビープを鳴らす関数である。pasteという定数名からもわかるとおり，Editメニューのペーストが選択されたときにこの行が実行される。

では，この行にブレークポイントを設定して，再びコンパイル&実行。が，Editメニューからペーストが選べないようになっているようだ。

ヒント6　ツールの機能を最大限利用する

どうも，Fileメニューの〝New〟では実行されている処理のどれかが，Editメニューのペーストでは実行されていないようだ。試しに，Cmd_Newという定数でソース内を検索してみよう。ObeyCommand()の中のほかに，FindCommadStatus()という関数の中でも使われているようだ。switch文の該当部分では，

```
outEnabled＝true;
```

となっている。ちょっとイタズラをして，ここを，

```
outEnabled＝false;
```

と書き換えてみよう。例によって，コンパイル&実行。さて，どうなっただろうか。今度は，Fileメニューの〝New〟が選択できない状態になっているはずである。

つまり，FindCommandStatus()という関数は，メニューの選択/非選択の状態を制御するために呼ばれる関数であるようだ。そこで，先ほど変更した部分を元に戻し，〝break;〟の後ろに次の3行を追加してみよう。

```
case cmd_Paste :
    outEnabled = true;
    break;
```

再コンパイル後実行すると，今度はペーストメ

ニューが選択できる状態になっているはずだ。

このように，不思議に思うことや，気づいたことがあったら，とにかくソースを検索してみることをおすすめする。CodeWarriorの検索機能は非常に優れており，複数のファイルに対して検索を実行できるし，検索結果のウィンドウで直接ソースを編集することもできるようになっている。

ヒント7 リファレンスツールを手に入れる

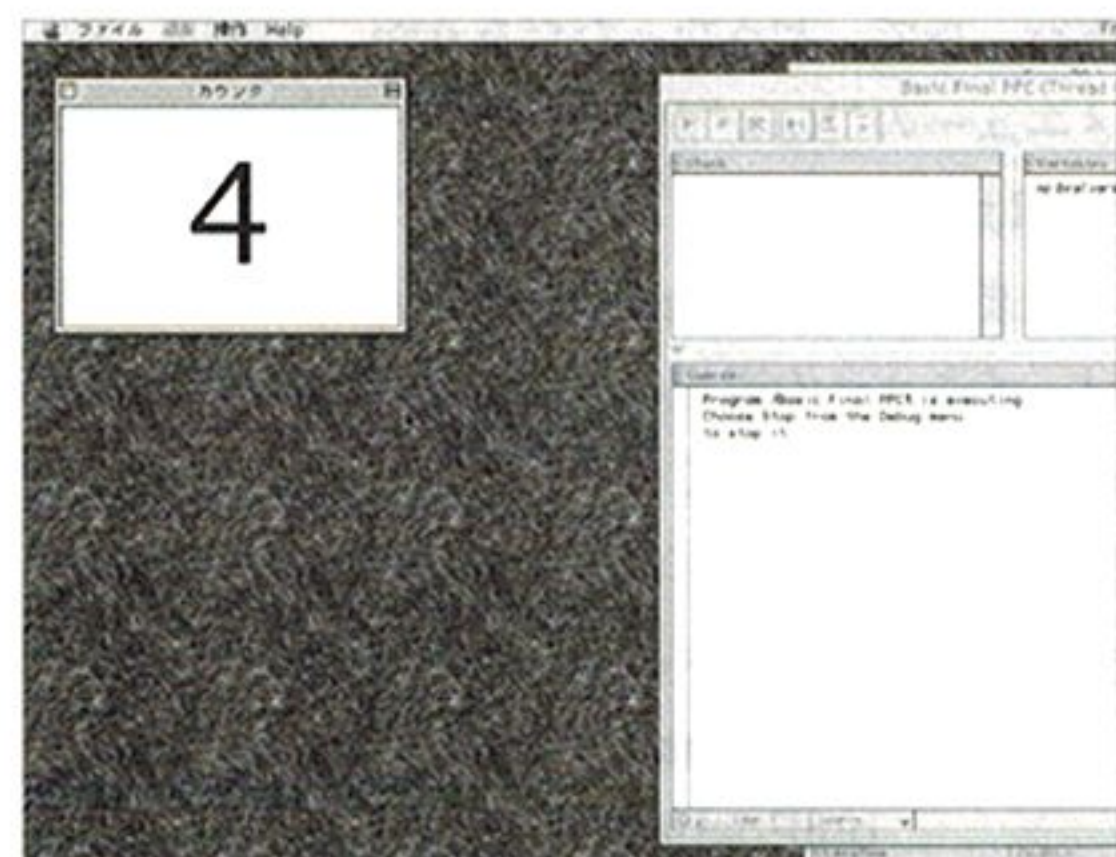
さて，そろそろサンプルプログラムに目を移そう。このプログラムは，メニューから選ばれた音を鳴らし，またメニューから項目が選択されたらウィンドウ内の数値を増減させる，という至極簡単なものである（画面5）。

224行からのPlaySound( )という関数を見てほしい（リスト1）。この関数は，ObeyCommand( )から呼ばれているのだが，渡されたリソースIDの音を鳴らす関数である。プロジェクトのリソースファイルを開くと，SNDという名前のリソースがあるが，このリソースファイルに登録されている指定番号の音を鳴らす関数だと考えていただければよい。

しかし，ここにきて急に見慣れない関数名が出てきた。GetResource( )，SndPlay( )などである。どちらも，MacOSのAPI関数で，名前のとおりの動作をする。

クラスライブラリがいくら網羅的に処理を請け負ってくれるといっても，音を鳴らすといった特殊なことをするためには，APIを直接使わなければならない。そのときに，目的の処理を実行するためにはどんな関数が必要で，その関数はどんな引数をとるかを調べるときに役立つのが，リファレンスツールだ。

リファレンスツールには何種類かあるが，まずいちばん手軽なのが，フリーウェアの"Toolbox Man"だろう。Vectorのサイト（http://www.vector.co.jp/）などからダウンロードできる。説明文が日本語なのがうれしい。基本的なAPIの仕様を調べるには，まずこれがあれば十分だろう。

もうひとつ，よく使われているリファレンスツールに，ToolBoxAssistantというものがある。これは，大きな書店などで売られている，Inside Macintosh CD-ROMなどについてくる。Macintoshが持つほぼすべてのAPIに関して記述されてあり，サンプルのコードなども充実しているのだが，記述が英語なので，英語が読める人にのみおすすめする。

画面5：サンプルプログラム
サンプルプログラムの画面。少ないカスタムコードでこれだけのことができるのがクラスライブラリの強み

リスト1　PlaySound( )のコード

```
//引数として与えられたResourceIDの音を鳴らします
OSErr
BasicPPApp::PlaySound(short inID)
{
		//音をリソースから読み込みます
		Handle sndH = ::GetResource('snd ', inID);
		if (sndH == NULL)
		{
				return ::ResError();
		}
		else
		{
				//鳴らす音のHandleをロックします
				StHandleLocker hlocker(sndH);
				//音を鳴らします
				return		::SndPlay(NULL,
(SndListHandle)sndH, false);
		}
}
```

リスト2　IsWindowThere( )のコード

```
//ウインドウがあるかどうかを調べ、Bool値を返します
Boolean
BasicPPApp::IsWindowThere()
{

		if( FrontWindow() )
		{
				return true;
		}
		else
		{
				return false;
		}
}
```

ここでは，とりあえず"Toolbox Man"を使ってみよう。サンプルのプログラムでは，ウィンドウはひとつだけ画面上に現れている。つまり，画面にウィンドウがある場合は，新規メニューを選択できない状態にする必要があるのだ。

CodeWarriorのマルチファイル検索を使って，"Toolbox Man"のファイルから「ウィンドウ」という文字が含まれている部分を検索する。すると，71件の検索結果が得られる。検索結果のなかからまず気になるのは，FromtWindow( )という関数。しかし，よく解説を読んでみると，ある点を与えて，その位置にウィンドウが存在するかどうかを探す関数らしいことがわかる。いまは単純に「あるかないか」を調べたいだけなので，この関数では役に立たない。

次に気になるのはFrontWindow( )という関数。これは，最前面にある ウィンドウを返す関数らしい。ウィンドウがない場合はNULLが返ってくるようなので，この関数の返り値を調べればウィンドウの有無が調べられるようだ。

そうしてでき上がったのがIsWindowThere( )という関数（リスト2）。メニューコマンドの有効/無効を決定するFindCommandStatus( )で利用している。

ヒント8 インターネットを活用する

プログラミング初心者のうちは，ひたすら手探りで突き進むしかないわけだが，ときとして，自分ひとりでは解決できない問題に突き当たることがある。ただ，自分が突き当たった問題は，たいてい他人も苦労した経験があるもので，また世の中には親切な方がおり，その手の問題の解決策をネット上で公開していたりするものである。

たとえば，検索サイトのgoo（http://www.goo.ne.jp/）で"Mac プログラミング入門"というキーワードで検索をかけると，入門者用の気の利いたサイトがいくつか見つかるはずだ。なかには，雑誌に連載された入門用の連載記事をWeb上で公開してくれているサイトなどもある。これらを活用しない手はない。

また，CodeWarriorのメーリングリストというものが存在する。日々，活発にMacintoshの開発に関する話題がやり取りされている。このメーリングリストは，ぜひとも購読しておきたい。

ヒント9 すでにあるものは活用する

CodeWarriorについてくるサンプルコードのほかにも，ネット上で公開されているMacintoshのサンプルコードというのはたくさん存在する。なにか，自分でやってみたいことができたら，とりあえず同種のことを実現しているサンプルコードがあるかどうかを調べてみることをおすすめする。ネット上に公開されている資産は，有効に活用したいものだ。

また，CodeWarriorを開発しているMetrowerksのサイト（http://www.metrowerks.com/）には，PowerPlant Contributed Classesというページがある。このページは，PowerPlantを利用し，特定用途に特化したカスタムクラスを集めており，多くのクラスがフリーで公開されている。検索機能もあるので，ここもぜひともチェックしておきたいもののひとつだ。

ヒント10 知っている人に聞く

なんだかオチのようになってしまったが，真実である。Macintoshのプログラミングに詳しい人物を見つけて，わからないことがあったら聞きまくる，というのが，上達へのいちばんの近道かもしれない。ただし，質問するのは，ひととおり自助努力をしてからにしよう。あまりタコな質問ばかり繰り返すと，人間関係を壊すおそれがある。たいていの人間の忍耐力には，限界がある。

最近は，電子メールという便利なものがあるし。え，ボクですか，いや，最近忙しいから……。でも，まあ，女性相手とかだったら……。ここまでどうぞ。→shibata@mail.at-m.or.jp

VisualC++で始めるWindowsプログラミング

菊地　功/Kikuchi Isawo

現在もっとも一般的なパソコンプラットフォームといえば，Windows95/98。もちろん，開発環境もいろいろ揃っている。そんななかでも，定番というか本道を行くのがVisualC++による開発だ。ここではVisualC++によるプログラミングの流れとWindowsの基本について見ていこう。

Oh!Xの読者だった人，お久しぶり。そうでない人，こんにちは。待ちに待っていたこのときが，やっとやってきました。誌名もそのままってのがちょっとすごいけど，また昔のようなヘビーな記事を書けるかと思うと，ちょっとげんなりです(あれ？)。ちょっとね，ここ数年お気楽なライター家業に甘んじてしまっていたので，結構不安はありますが，はりきって行ってみましょう。

さて，本誌を初めて手にした人は，ぱらぱらとページをめくってすでにちょっと戸惑っているかもしれません。そう，この本は「開発言語くらい知ってるだろ」って前提で，ばりばりとプログラムのソースコードリストが登場します(編注：いや，そんなには入れないです)。しかも，Windowsになって，ただ言語の仕様を知っていればいいというわけではなくなってきました。

というのは，開発環境によって，それぞれ操作性が異なるからです。Windowsでの開発環境といえば，BASIC系(VisualBasic)，Pascal系(Delphi)，C++系などがあります。人によってVisualBasicはお手軽だとか，Pascalは美しいとかあるでしょうが，C言語使いの筆者としては，当然のようにC++をメイン(というか，それしかわからん)としています。C++を使うときの最大のメリットは(アセンブラはおいといて)，なんでもできるという点でしょう。「高級言語の皮をかぶったアセンブラ」などとよく称されますが，アセンブラでできることは(少々遠回りになっても)すべてできるわけですから，いちばんマシン語(って死語か)に近い高級言語といえます。いちばんネイティブに近いともいえるでしょう。つまり，一般的にいちばん高速なコードを吐けるわけです。実際に，ゲーム業界でもいまではC++(あるいはC)でのプログラムが一般的となっています。

C++といっても，Windows環境ではいくつかの開発環境が発売されていますが，私はマイクロソフト純正のVisual C++(以下VC++)を使っています。私はこれしか使ったことがないのですが，「できることならマイクロソフト製品は使いたくない」という人もいるかもしれません。気持ちはわかります。私とてマイクロソフトの信者ではありませんので。しかし，VC++を使っている限り，新しいWindowsの機能などはきっちりサポートされますし，DirectX SDKもVC++ならばそのまま使うことができます。こう考えるのはどうでしょう。「利用できるものは利用する」それでもやっぱりインプライズがいいとかいう人に無理矢理VC++を使わせる気はありませんが，基本的に本誌で取り扱うC++環境はVC++になると思いますので，それぞれで苦労してください。

というわけで，VC++の使い方は自分で勉強しろ，というのはあんまりですので，一発目としてVC++の使い方についてザーッと解説します。

まず必要なのは，Windowsが走るマシンとVisual C++。マシンはインストールできるHDDのスペースさえあれば，CPUはいくら遅くてもコンパイルできないということはありません。が，コンパイルに時間がかかっていてはプログラミングの思考が中断されてしまいますので，できるだけ速いものが好ましいです。「開発とゲームは速いマシンでやれ」という格言があるとかないとか。副作用としては遅いマシンではまともに動かないアプリケーションができる可能性があるというくらいです。

Visual C++のほうは，バージョンは現在5.0ですので，それをベースとします。基本的に4.xとできることは同じですが，ActiveXなどが作れるようになっていることと，環境がほかの言語と統一されてDeveloperStudioに載っていますので，操作性は若干異なります。

さて，VC++5.0には3つのエディションがあります。

Learning Edition
Professional Edition
Enterprise Edition

Learning Editionはその名のとおり，学習用です。最適化が弱かったり，作ったプログラムを販売してはいけないという制約がありますが，約2万円(オープンプライス)ほどで購入できます。できることならばProfessional Editionがほしいところですが，こちらはグッと値が跳ね上がり，6～7万円ですので，最初はLearning Editionで我慢するのが正解かもしれません。おそらくバージョンアップ時にProfessional Editionに乗り換えることができると思いますので，慣れてきたらそちらに移行するというのも手です。Enterprise Editionはさらにサーバーアプリケーションの開発やグループ開発などをサポートしていますが，個人レベルでは必要ないでしょう。

また，VC++はVisual Studioという，複数の開発環境がパッケージされた製品にも含まれています。こちらはProfessional EditionとEnterprise Editionがあり，それぞれVC++のProfessional EditionとEnterprise Editionが入っています。さらに，VisualBasicやVisual J++などの開発環境も入っており，すべてをバラで買うよりもお得ですので，必要な人はそちらを買うとよいでしょう。なお，筆者はマイクロソフトの些細な陰謀にあい，Visual StudioのEnterprise Editionを所有しています。

■CからC++言語へ

Visual C++の製品群について説明したところですが，それ以前にまずC++言語についての知識がないと始まりません。ただ，基本から全部説明していては本誌が電話帳並みになってしまいますので，C言語の基本は知っているものとして，C++で新しく追加された部分について簡単に説明するにとどめましょう。C言語があやふやだという人は，コラムに基本的なリファレンスを載せておきますので，そちらを参考にしてください。それでもなにがなにやらさっぱりという人は，専門書でも買って読んでください。

ただ，筆者の経験からいって，こういうものは学習よりも実践です。最初はよくわからなくても，プログラムリストを眺めているうちに，わかってくるものです。

さて，C++で拡張された機能でいちばん大きいのはなんといってもクラスですが，まずは細かいところからいってみましょう。

●ファイル拡張子

ヘッダファイルの拡張子はCと同じhですが，ソースファイルの拡張子はcppです。

●関数宣言

Cで関数を定義する場合，

```
func (a, b )
int a;
int b;
{
    :
}
```

などとできましたが，C++では，

```
func (int a, int b )
{
    :
}
```

という記述法が推奨されています。上の方法でもコンパイルはできますが，下で統一するようにしましょう。

●多重定義

同じ文字列(記号)で関数や演算子を宣言できます。たとえば，

```
void func (int i ) ;
void func (char *s ) ;
```

という2つの関数を定義しておけば，プログラム中から，

```
func( 10 ) ;
```

とすれば上の関数が，

```
func( "Oh!X" ) ;
```

とすれば下の関数が呼ばれます。また，C++の解説書の中で，

```
cout << "Hello World!";
```

などといった記述を目にすることがあるでしょう。ここで，coutというのはコンソール出力のストリームI/Oで，もともと左シフト演算子である'<<'は，coutに対しては「ストリームに出力しなさい」という多重定義がなされているのです。つまり，上の例では，標準出力に"Hello World!"という文字が表示されます。ただ，実際問題としてこの演算子は，「多重定義ができますよ」というサンプルみたいなもので，とりわけコンソール出力なんてWindowsでは意味がないので，忘れて構いません。

●ローカル変数の宣言

Cでは関数中のローカル変数は関数の先頭でしか宣言できませんでしたが，C++ではどこででも宣言できます。ただし，当然のことながらその変数は宣言した以降で，しかも'{…}'で括られた複文中で宣言した場合は，その中でしか有効ではありません。こういった「変数が有効な範囲」をスコープといい，とりわけいまのような変数をローカルスコープといいますが，別に難しい話ではなく，当たり前のことです。ちょっと紛らわしいのは次のような場合です。

```
for (int i=0; i<10; i++ ) {
    int j;
    :
}
```

for文の初期化式で変数iが宣言されていますが，この場合はfor文の中だけでなく，それ以降も変数iは生きています。それに対し，複文中で宣言された変数jは，forループを抜けると無効になります。さらにいえば，変数jはループ1回ごとに生成と消滅を繰り返していますので，変数jを用いて次のループに値を持ち越すのは危険です(常にスタックの同じ位置に配置されれば，以前の値を引き継いでいるように見えるかもしれませんが)。

●コメント

Cではコメントは'/*'と'*/'で括られた範囲でしたが，C++では'//'から行末(改行コード)までもコメントとみなし，コンパイル時に無視します。これはANSIではC++から規格化されたものですが，一部のCコンパイラではサポートされていましたので，知っている人も多いでしょう。

●new,delete演算子

ヒープからメモリを確保する演算子と，それで確保したメモリを解放する演算子です。次のように使用します。

```
int *i = new int;
    :
delete i;
```

C言語を知っている人は「はぁ？」と思うでしょう。おおよそCらしくない演算子です。malloc( )を使った場合の，

```
int *i = (int *) malloc ( sizeof (int) ) ;
    :
mfree( i );
```

と同じだと思って構いません(たぶん)。int変数ひとつを確保するだけではあまりうれしくありませんが，もちろん配列も確保できます。

```
int *i = new int[n];
    :
delete[ ] i;
```

ますますらしくないですが，要素数nは変数でも構わないというのがありがたいところです。ただし，配列の場合にはdeleteの後ろに'[]'をつけます。忘れてもエラーにはならないのですが，おそらくメモリにゴミが残ることになるので注意しましょう。

●デフォルト引数

関数の引数にデフォルト値を指定し，そのデフォルト値そのままでよい場合には，関数呼び出し時に引数を省略できるようになりました。たとえば次のように指定します。

```
void func( int a, int b=10 ) ;
```

これは必ず最初の宣言時にのみ行います。プロトタイプ宣言をしている場合にはその宣言時だけで，関数の定義は通常どおりになります。このように宣言することで，

```
func( 5 ) ;
```

として呼び出した場合，

```
func( 5, 10 ) ;
```

とした場合と同様になります。ただし，デフォルト引数は後ろのほうの引数にだけ指定が可能ですので，

```
void func( int a, int b=10, int c=2 ) ;
```

とはできても，

```
void func( int a, int b=10, int c ) ;
```

とはできません。

●参照演算子

関数の宣言および定義時に，

```
void func( int &a ) ;
```

として変数型に'&'をつけることで，引数の変数がアドレス渡しになります。上の関数が，

```
void func( int &a )
{
    a += 1;
}
```

として定義されており，ほかから，

```
func( i ) ;
```

として呼ばれた場合(iはint型)，結果としてiには1が加算されます。これをもし参照演算子を用いなかった場合には，iの値をスタックに積んで(コピーして)func( )関数を呼ぶだけですから，i自体の値は変化しません。上と同じ機能を参照演算子を使わずに実現するには，

```
void func (int *a )
{
    *a += 1;
}
    :
func( &i ) ; // ←これはアドレス演算子
```

となります。

Cとの違いはだいたいこんなところです。あとはあったとしても筆者が気づかない程度の有用性の低いものでしょうから，実用ではこれくらいを押さえておけば大丈夫でしょう。

■クラスとはなにか

さて，いよいよクラスの登場です。

ところで，クラスはC++だけでなく，概念自体はVisualBasicやDelphi，あるいはJava(JavaはC++をベースとしていますが)にもあります。確かにオブジェクト指向とか，大規模開発に向いているという話もありますが，Windowsでの開

発に関しては，もう少し密接な関係があります。

Windowsの画面は，デスクトップがあり，その上にタスクバーやアイコンが散らばっていて，さらにアプリケーションが開いていたりしますが，こういったものはすべて「ウィンドウ」でできています。ダイアログはもちろん，デスクトップだってタスクバーだって，タスクバーに乗っかってるスタートボタンもウィンドウなのです。目に見えていてウィンドウじゃないものは，カーソルくらいなものです。

一般的にいって，ウィンドウといえばタイトルバーがあって，左上にアイコンと，右上にクローズボタンがあったりしますが，それもウィンドウ，デスクトップもウィンドウ，ボタンもウィンドウ，なぜ同じウィンドウがこんなに姿を変えてしまうのでしょうか。

実は本来のウィンドウというのは，座標や表示状態など，必要最低限のパラメータしか持っておらず，そのままでは姿すらないのです。そこからさまざまなものを「派生」することによって，あるものはタイトルバーを持ったり，あるものはボタンになったり，ボタンになったものでも，表面にテキストが表示されるものとビットマップが貼り付けられるものに分かれたりと，さまざまなウィンドウ形態を持つようになるのです。さらに親子関係や属性の継承など，クラスというのは，このWindowsのシステムを記述するのに最適な概念なのです。

抽象的な話から具体的な話に移りましょう。とりあえず派生については置いといて，クラスというものをちょっと乱暴にいってしまうと，「関数をメンバに持てる構造体」と説明できます。構造体はなんらかのまとめられるパラメータをパックしたものですが，クラスはそれらのパラメータをどう料理するかといったレシピまでパックできるわけです。

まずはここが「オブジェクト指向」といわれる由縁です。要はそのクラスをライブラリ化してしまえば，その中でなにが行われるかを知ることなしに，仕様書どおりに関数を呼んでやれば出力が得られるわけです。クラスの宣言は次のようにします。

```
class CMyClass
{
public:
        CMyClass ( ) ;
        ~CMyClass ( ) ;
        void SetParameter ( int i ) ;
        int GetParameter ( ) ;
protected:
        int m_Param;
};
```

ヘッダファイルの中で上のように宣言し，ソースコードの中では次のようにメンバ関数（メソッド）を定義します。

```
CMyClass::CMyClass ( )
{
        m_Parame = 0;
}
CMyClass::~CMyClass ( )
{
}
void CMyClass::SetParameter ( int i )
{
        m_Param = i;
}
int CMyClass::GetParameter ( ) ;
{
        return m_Param;
}
```

このように定義できれば，プログラム中から次のように利用できます。

```
CMyClass ＊pmyclass = new CMyClass;
pmyclass->SetParameter ( 10 ) ;
        ⋮
int i = pmyclass->GetParameter ( ) ;
delete pmyclass;
```

それではこのクラスについて説明していきましょう。

まずはクラスの宣言から。publicとprotectedというキーワードがあります。これはアクセス権を指定するもので，その関数なり変数が外部からアクセスできるかどうかを示しています。「外部」というのはピンとこないかもしれませんね。つまりそのクラス内のメソッドを「内部」とした場合の，それ以外のコードからのアクセス権についてです。publicがアクセス可，protectedは不可を示し，このキーワードは次のキーワードが出るまで有効ですので，上のクラスではCMyClass ( )メソッドからGetParameter ( )までがパブリックメンバ，m_Param変数だけがプロテクトメンバということになります。ですから，クラスメソッド外のプログラム中から，

```
pmyclass->m_Param = 10;
```

ということはできず，コンパイル時にエラーとなります。いわゆるプロテクトメンバは「ブラックボックス部分」ということになります。ここではたまたまメソッドがすべてパブリック，変数がプロテクトになっていますが，もちろんプロテクトメンバのメソッドや，パブリックな変数も宣言できます。また，このほかにprivateといったキーワードもありますが，それはまたのちほど。

さて，クラスというものは親切なことに，インスタンス（メモリ上に割り当てられたクラスの実体）を生成したときと，インスタンスを廃棄するときに，特定のメソッドが呼ばれるようになっています。これを「コンストラクタ」と「デストラクタ」といい，コンストラクタはクラス名と同じメソッド，デストラクタはクラス名の前に‘~’をつけたメソッドと決まっています。この場合はCMyClass ( )がコンストラクタ，~CMyClass ( )がデストラクタで，それぞれ，

```
CMyClass ＊pmyclass = new CMyClass;
```

と，

```
delete CMyClass;
```

のタイミングで勝手に呼び出されているのです。コンストラクタでは初期化処理を，デストラクタでは後始末をするのが一般的です。ここではコンストラクタでメンバm_Paramを初期化し，デストラクタでは特になにもせずに終了しています。そうそう，メソッドの前の‘CMyClass::’は，この関数がCMyClassのメンバであることを示しています。

次の，

```
void SetParameter ( int i );
```

メソッドは，内部的にはプロテクトメンバm_Paramに値を設定しているだけです。先ほどもいったようにm_Paramは外部からは直接アクセスできないので，値を設定するにはこのメソッドを，取得するにはその次の，

```
int GetParameter ( ) ;
```

メソッドを呼ばなくてはならないわけです。こういったメソッドを呼ぶには，構造体と同じように，メンバ参照演算子‘.’，あるいはインスタンスへのポインタの場合は‘－>’を使います。いまはpmyclassがポインタですので，

```
pmyclass->SetParameter ( 10 );
```

となりますが，これを無理やり‘.’を使えば，

```
(＊pmyclass).SetParameter ( 10 );
```

とも記述できます。ちなみに内部からアクセスする場合は，SetParameter ( )メソッド内にもあるように，単にm_Paramと記述すれば，それが自分のクラスのメンバであると判断されます。この例ではパラメータ1個をクラス内に保存しているだけですのでたいした意味はありませんが，だいたいのイメージはわかっていただけたんじゃないでしょうか。では次，このクラスを派生させてみましょう。

```
class CMyNewClass : public CMyClass
{
public:
        CMyNewClass ( int mul ) ;
        void SetParameter ( int i ) ;
protected:
        int m_Multiple;
};

CMyNewClass::CMyNewClass ( int mul )
{
```

```
        m_Multiple = mul;
    }
    CMyNewClass::SetParameter ( int i )
    {
        m_Param = i*m_Multiple;
    }
```

クラス名の後ろ':'に続く'public CMyClass'というのは，先ほどのCMyClassを派生させるよ，という宣言です。

ここでいう派生とは，CMyClassのメンバすべてを引き継ぎつつ，次に宣言するメンバを追加するという意味です。

さて，今度はコンストラクタが引数を取っています。このように初期化時に必要なパラメータはコンストラクタが受け取ることができ，プログラムでは次のように記述します。

```
    CMyNewClass ＊pmynewclass =
        new CMyNewClass ( 2 );
```

このようにインスタンスを生成することで，'2'がコンストラクタに渡され，プロテクトメンバm_Multipleが'2'で初期化されます。

その後ろには，CMyClassでも宣言した，

```
    void SetParameter ( int i );
```

がまた出てきています。これはオーバーライドといい，前のクラス（スーパークラスといいます）で宣言されたSetParameter ( )メソッドは新しいメソッドと入れ替えられ（引数が異なる場合は多重定義となります），以降またオーバーライドされるまで派生クラスでもこちらが有効となります。

要は，SetParameter ( )メソッドに新しい機能を追加したわけです。ここではスーパークラスのSetParameter ( )メソッドがプロテクトメンバm_Paramに渡された値を代入しているだけと知っているので，今回はコンストラクタで渡されたm_Multipleを掛け合わせてm_Paramに入れているだけですが，もしスーパークラスがなにをしているのかわからないという場合は，

```
    CMyNewClass::SetParameter ( int i )
    {
    CMyClass::SetParameter ( i＊m_Multiple );
    }
```

として明示的にスーパークラスのメソッドを呼ぶこともできます。コードを見ればなにをしているかわかると思いますが，CMyClassでは単に与えられた数値を保存するだけでしたが，派生したCMyNewClassはインスタンス生成時に渡した倍率で数値を保存するというように「進化」しているわけです。

だいたいわかってもらえたでしょうか？　え，わかった？　ほんとに？　嘘ですね。じゃあ，

```
    class CMyNewClass : public CMyClass
```

の'public'ってなに？　さっきはさらっと流したけど，こいつはね，継承時のメンバのアクセス権を指定しているんですよ。

publicならばスーパークラスのパブリックメンバもプロテクトメンバも，そのままパブリックとプロテクトで継承されます。しかし，protectedにすると，パブリックもプロテクトになっちゃいます。

あと，もうひとつprivateってのがちらっと出てきたけど，これは宣言した世代しかアクセス権がなく，派生したクラスからもアクセスできなくなるメンバ。しかも継承時にprivateを指定すると，パブリックメンバもプロテクトメンバもプライベートメンバになってしまいます（表1）。なんて偉そうにいってみたけど，実際にはpublic以外を使うことはほとんどないでしょう。あと，friend classというちょっと特殊なものもありますが，private以上に使うことはまれですので，説明は省略します。筆者も使い方（使い道）がいまいちよくわかりませんし。

最後に，ちょっと重要な「thisポインタ」について話をしておきましょう。これはクラスのメソッド内だけに存在するポインタで，「自分自身」を指し示しています。つまり，メソッドが呼ばれている以上，そのメソッドがいるクラスのインスタンスは生成されているわけで（まれにそうでない場合もありますが），そのインスタンスへのポインタを指しています。いままでの例でいえば，pmyclassなりpmynewclassの値そのものをthisで取得することができます。

たとえば，SetParameter ( )メソッド内からm_Paramをアクセスする場合，そのままでよいといいましたが，正確にはthis.m_Paramとなり，'this.'に限って省略しても構わないということなのです。

では，このthisポインタはどこからきているのでしょうか。もちろんメンバではありません。実は，メソッドの呼び出し時に「こっそり」引数として渡されているのです。第0引数とでもいいましょうか，C言語風に書けば次のようになっているのです。

```
    SetParameter ( CMyClass ＊this, int i )
```

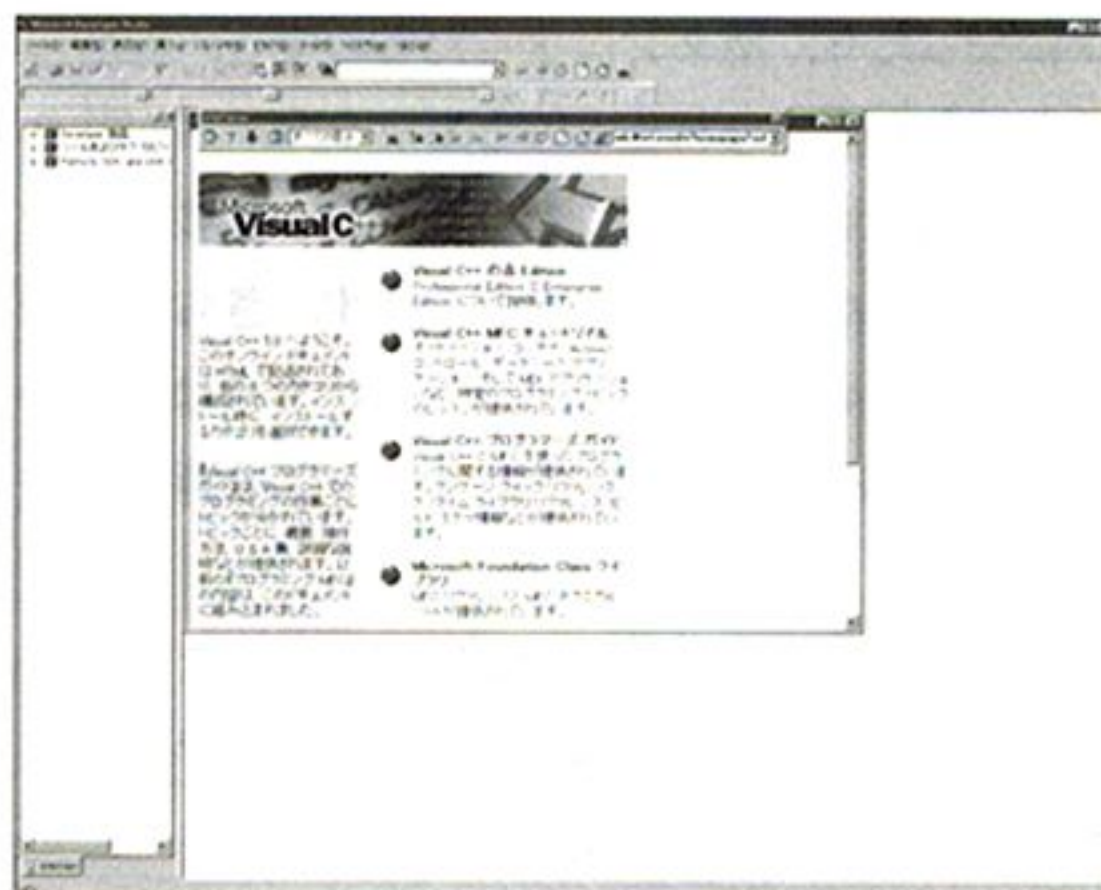
しかし，C++でクラスを使っている限り，その辺りはうまい具合に覆い隠され，あたかもthisポインタは突然湧き出したかのように見えるのです。

このthisポインタ，いまはどのように使うのか実感が湧かないと思いますが，結構頻繁に使いますので，覚えておいてください。キーワードは「困ったときのthisポインタ」です。

■Visual C++を立ち上げよう

大丈夫ですかぁ？　ちゃんとついてきてますかぁ？　やっとこさ実技に入りますよぉ。筆者としてはこういう記事は久しぶりなのでうきうきしてるわけですが，読者はきっとたいへんだろうなぁ，と思いつつも，改心の兆しもなくさくさくと先に進んでしまいます。

インストール作業とかはVisual C++関係のマニュアルにでも任せるとして，さっそくVC++（というかDeveloper Studio）を立ち上げてみてください。とりあえず最初に表示されるワンポイントなんてのはさくさく閉じちゃって，InfoViewトピックも邪魔なんで消しちゃいましょう（図1）。

図1　VC++を起動

さて，この広大なMDIの作業領域を見て，まずなにをしていいやら途方にくれることでしょう。

あるいは，プログラム経験のある人ならば，なにはともあれ新規ファイルを作成して，おもむろに，

```
    #include <stdlib.h>
    #include <stdio.h>

    void main ( int ac, char ＊av[ ] )
            ⋮
```

なんて書き始めるかもしれません。ちょっと待った待った，VC++では（というか最近の開発環境では），まずプロジェクトというものを起こすのですよ。とにかくまずは次のようにやってみてください。

［ファイルメニュー］の［新規作成］を選択

↓

［プロジェクト］タブが開いているはずなので，［MFC AppWizard (exe)］を選択し，プロジェクト名に"test"とでも入力してOKする（図2）

表1　アクセス権の継承

スーパークラスでのアクセス	スーパークラスからの継承	派生クラスでのアクセス
パブリック プロテクト プライベート	パブリック	パブリック プロテクト アクセス権なし
パブリック プロテクト プライベート	プロテクト	プロテクト プロテクト アクセス権なし
パブリック プロテクト プライベート	プライベート	プライベート プライベート アクセス権なし

図2　MFCを使ったEXEを指定

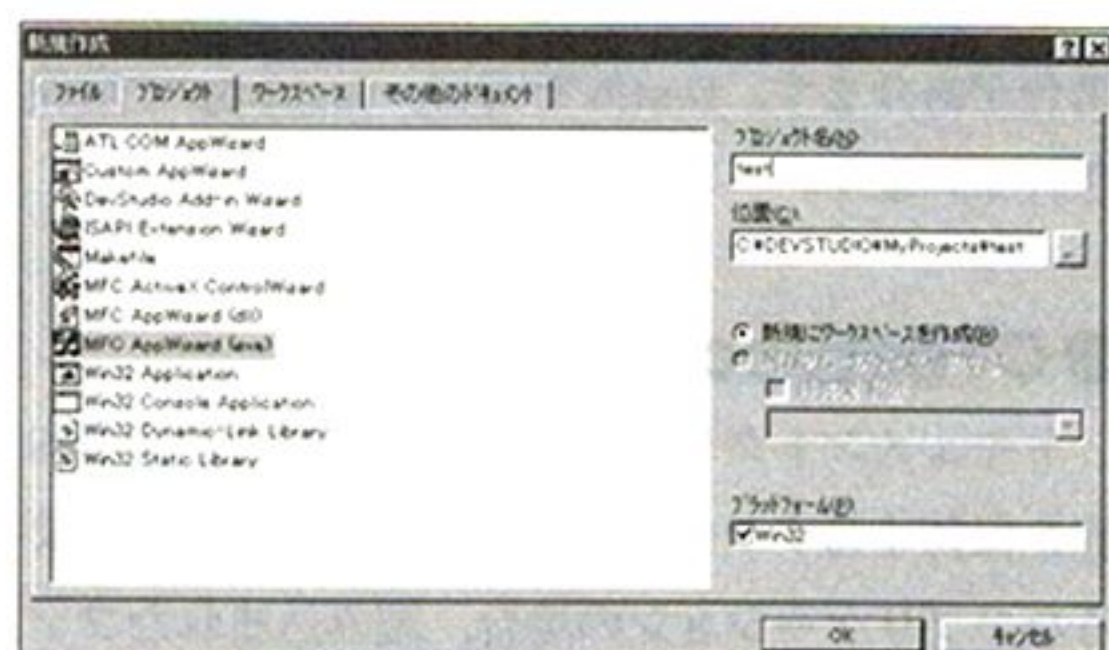
↓

作成するアプリケーションの種類をSDIにして次へ（図3）

図3　SDIアプリを

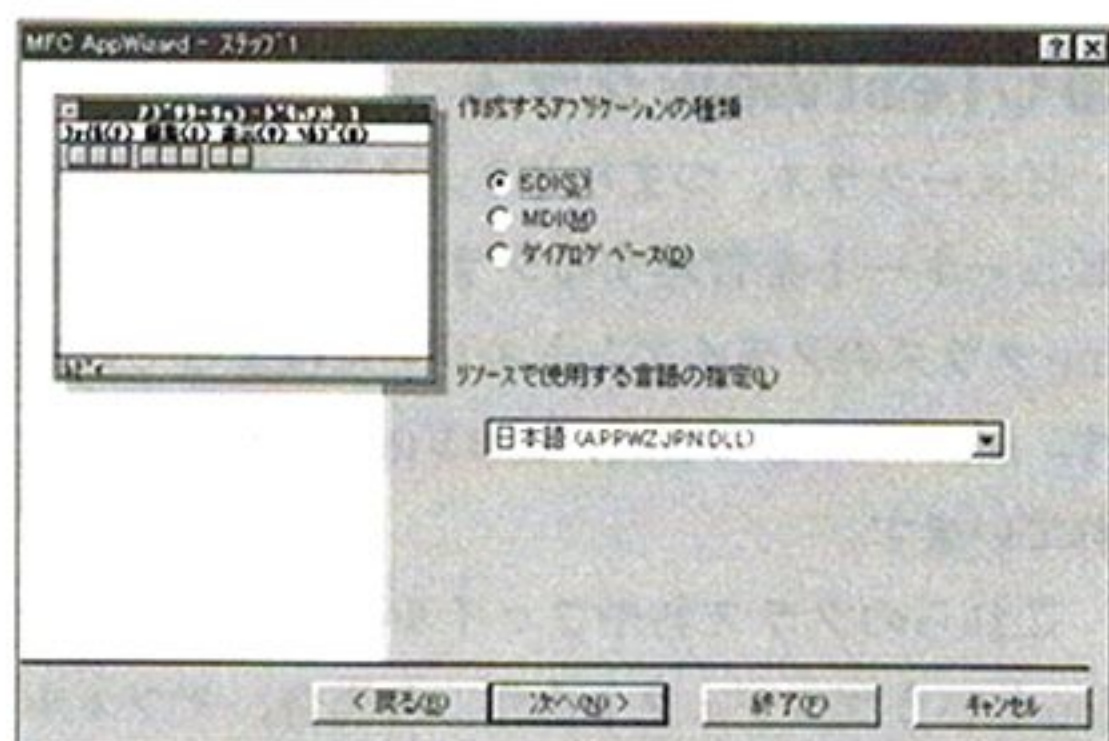
↓

データベースのサポートは「しない」のまま次へ（図4）

図4　とりあえずそんなものは不要

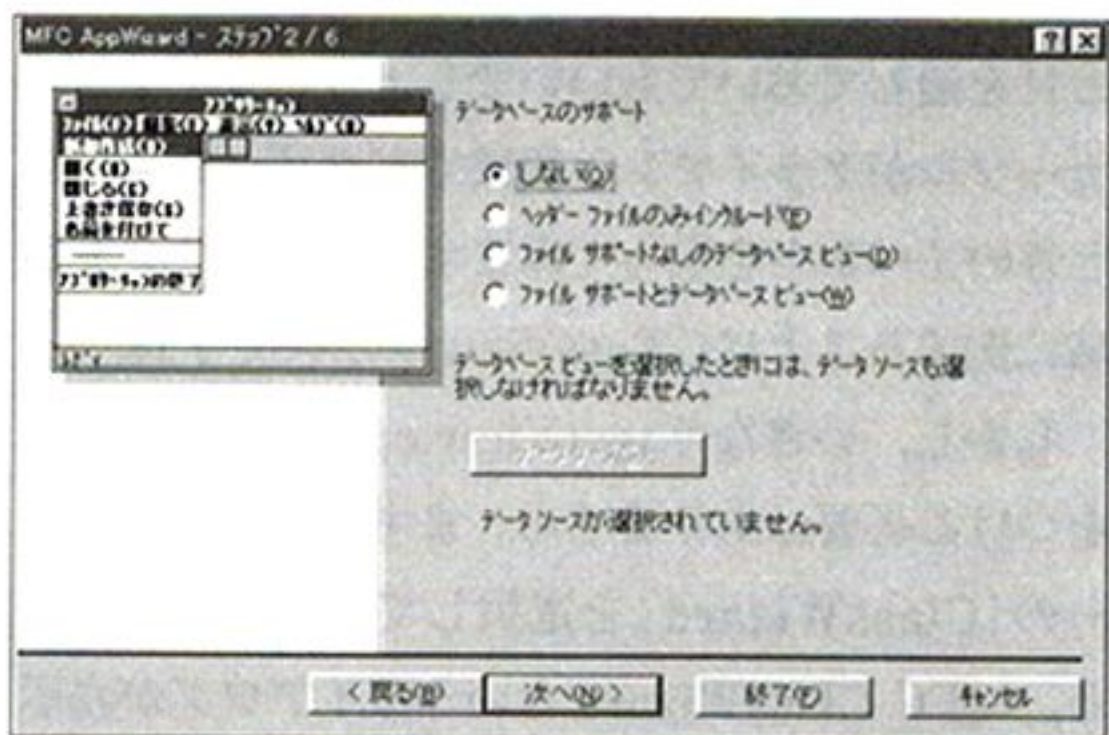
↓

さらに気にせず次へ×2（図5，6）

図5　当分気にしなくてもよい

↓

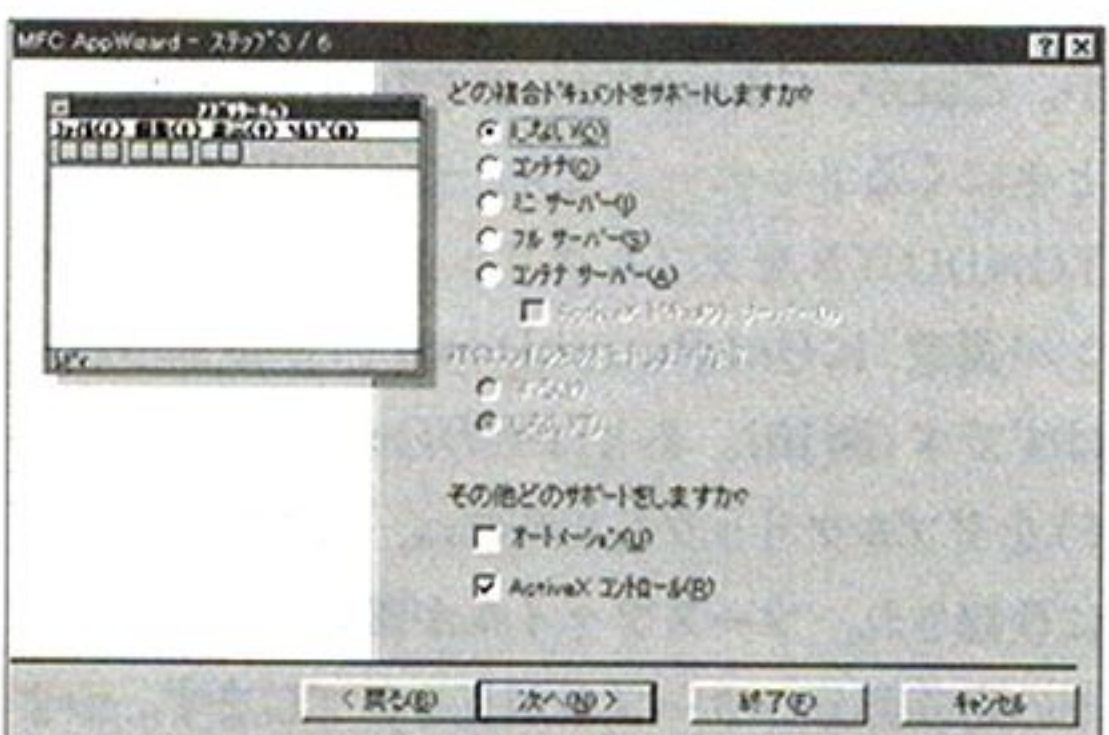



図6　ごく普通のものでいい

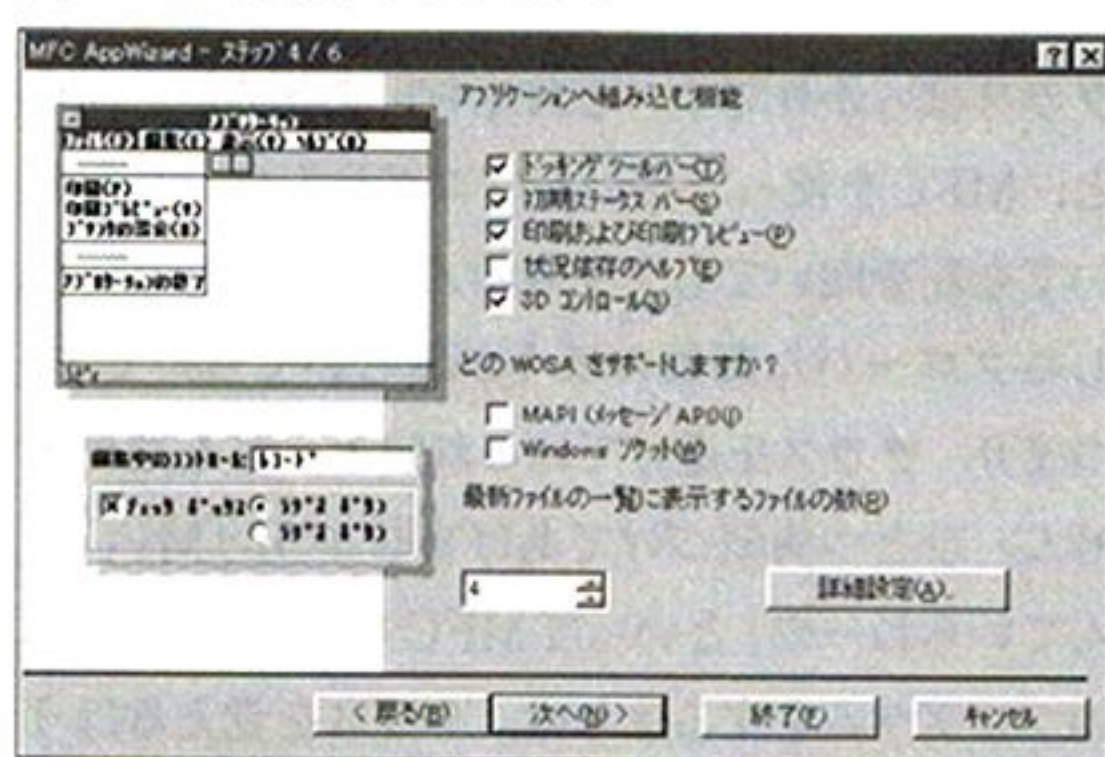
↓

ステップ5で「MFCのスタティックライブラリを使用」をチェック（図7）

図7　共有DLLでも可

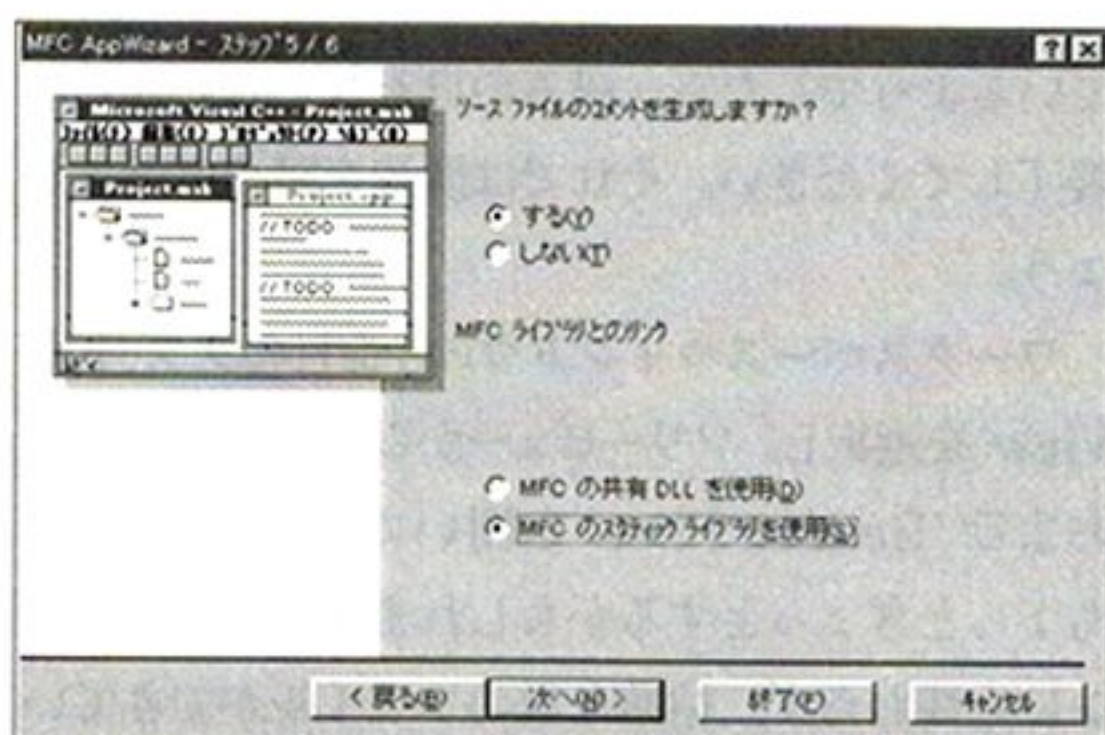
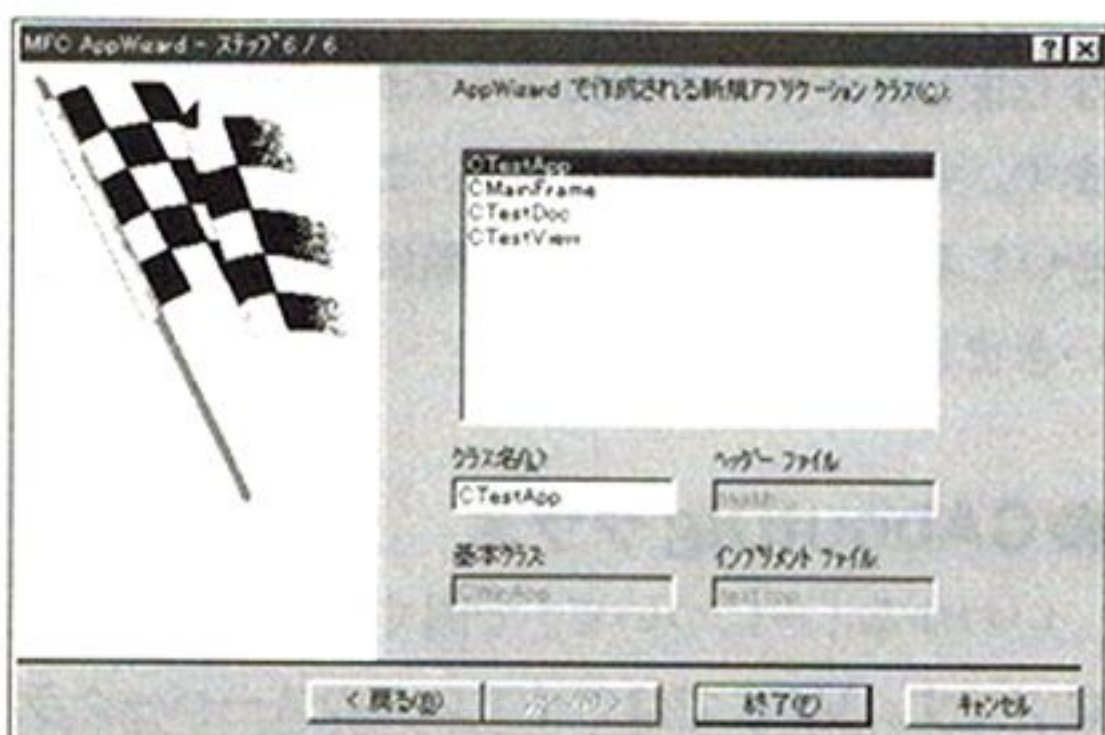
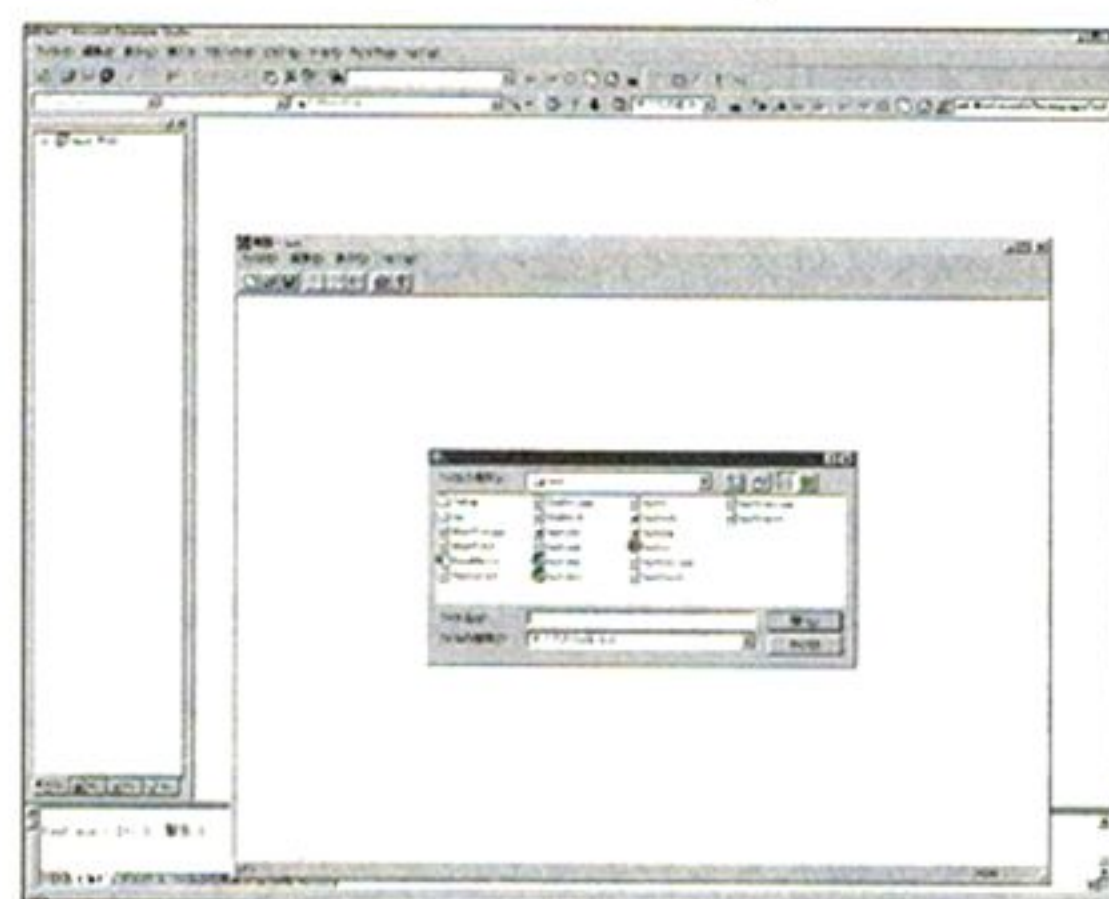
↓

最後は適当に眺めて終了（図8）

図8　これで終了

新規プロジェクト情報というダイアログが出ますが，それもOKしてください。すると，HDDがちょっとゴロゴロと音がして……なにも変化がない。いえいえ，ちゃんと変わっています。画面左の縦長のウィンドウ（ワークスペース）の下にタブが増えているでしょう。とまあそれを確認したら，おもむろにF7キーを押してみてください。突然画面下にウィンドウ（アウトプット）が現れて，ゴロゴロいい出しましたね。F7キーはビルドのショートカットキーですので，覚えましょう。

キーを打ってプログラムしたあとに，［ビルド］メニューを開いて，［ビルド］を実行……なんてやるよりずっと速いです。「ってまだなにもソース書いてねーよっ！」まあまあ，慌てなさんな。と

図9　なにもしなくてもこれだけ動く

にかくビルドが終わるまでじっと待ってください。アウトプットウィンドウに，

　test.exe - エラー 0，警告 0

と出ればビルド終了です。

さらにおもむろにCtrl＋F5を押してください（実行のショートカットです）。すると……おお！　なにもしてないのにウィンドウが！（図9）　しかもメニューも動作するし，［ファイル］メニューの［開く］でファイルダイアログが開く！　でも開いてもなにも起こらない！　そりゃまだなにもインプリメントしてませんから。要するに，ここまでをスケルトンとして勝手に作ってくれるわけです。

唐突ですが，ちょっとWindowsのアプリケーションの仕組みを説明しておきましょう。Windows自体はもちろんマルチタスクOSですが，個々のアプリケーション自体は基本的にイベントドリブンで稼働しています。つまり，マウスが押されたとか，画面を更新しなさいとかいうメッセージがOSからアプリケーションにくると，まずはそのメッセージはアプリケーションごとに用意されたメッセージキューに溜められます。そして，アプリケーションは「暇なとき」にそのキューをチェックし，もしメッセージがきていたらそのメッセージを振り分けて，それに対応した処理を行うわけです。これをメッセージポンプといいますが，本来はこういった面倒臭いこともユーザーがやらなければなりません。

さらに，ウィンドウを表示するにはまずウィンドウクラスというものをシステムに登録しなければなりませんし，表示だけでもいろいろなメッセージを処理する必要があり，面倒臭いことこの上ありません。では，まだなにもしていない先ほど作ったtest.exeは，なぜ動いているのでしょう。ウィンドウが表示されるのもそうですし，メニューが開くのも正常にメッセージを取得できているからにほかなりません。実は秘密はプロジェクトの種類にあります。

先ほどはMFC AppWizard（exe）というオブジェクトを起こしました。MFCというのは，Mic

rosoft Foundation Classの略で，このMFCライブラリはそういったWindowsプログラミングの泥臭いところを一手に引き受けてくれるありがたいライブラリなのです。これのお陰で，スケルトンを生成しただけで，あたかもいっぱしのWindowsプログラマになったかのような気分を味わえるのです。

それだけではありません。MFCライブラリにはプログラムにおいて便利なクラスがてんこもりなのです。ただ，ありがたいばかりかというと，残念ながらそうでもありません。まずこういったものの基本ですが，範囲外のことをやろうとすると，突然面倒なことになるということです。特に，土台に覆いをかぶせてまったく見えなくしているわけですから，その辺りに手を出そうとすると，かなり厄介な話になります。

もうひとつの問題は，サイズがむやみに大きくなるという点です。先ほどはデバッグモードのままビルドしたので巨大ですが，リリースモードでビルドしてもスケルトンだけで300KBを超えてしまいます。これをもしMFCライブラリを使わずに作れば，おそらく数十KBでしょう。こういったファイルの巨大化を防ぐために，MFCには共有ライブラリというものもあります。これは，MFCライブラリをDLLとして外部に追い出してしまい，複数のアプリケーションからこのDLLを共有しようというものです。プロジェクトを生成するときの，MFC AppWizard ステップ5の［MFCの共有DLLを使う］というのがそれです。これを指定すると，アプリケーション本体のサイズは小さくなりますが，このアプリケーションを受け取ったユーザーは，実行するために別途DLLが必要になります（DLLをアプリケーションに添付することは許されていますが，ますますサイズが大きくなります）。

筆者はほかのファイルに依存するというのが生理的に気に入らないので，今回のようにスタティックライブラリを使うようにしていますが，これは好きなほうを使って構いません。どちらを選択しても，ユーザーが書くプログラム自体はまったく変わりません。ただ，スタティックライブラリのほうが，DLLをコールするオーバーヘッドがない分，高いパフォーマンスを示す可能性があることを付け加えておきます。

MFCそのものを使わないようにするには，新規作成時にWin32 Applicationなどを選択すればいいのですが，こちらはスケルトンなどをまったく世話してくれません。筆者は簡単なものを作るときはMFCを使わないようにしていますが，ここでは入門ということでMFCを使う場合のみの説明にとどめておきます。

さて，ここへきていやーな予感がする人も多いかもしれません。C++の解説を読んで「別にクラスなんて使わなけりゃいいや。構造体もそれで切り抜けたし」と思ってた人。Microsoft Foundation Classライブラリというくらいです。もちろんクラスばっかりです。どれくらいクラスかというと，MFCを使って書いたプログラムは，右を向いても左を向いてもクラスのメソッドばかり。むしろグローバル関数のほうが珍しいくらいです。つまり，クラスを避けてはこれ以上一歩も進めません。ひとつ勇気づけてあげましょう。「MFCなんていらねーや」といってMFCなしでWindowsのプログラムをするよりは，クラスを覚えるほうが簡単です。

■プログラムの構造

話がそれましたが，先ほどのプロジェクトtestに戻りましょう。とりあえずtest.exeがまだ動いているようでしたら，浸るのはそれくらいにして終了してください。それでは構造を眺めてみましょう。

ワークスペースウィンドウの3番目のタブ‘File View’を選択し，ツリービューから‘Source Files’および‘Header Files’を開いてみてください。ちょっとぎょっとするかもしれません。スケルトンにも関わらず，結構な数のファイルができています。まだ投げないでくださいね。どれがなんのファイルかわかれば，それほど難しくはありません。では今度は1番目のタブ‘Class View’を開いて，ツリービューを開いてください。泣きたくなりましたか？　すでにクラスが5個もできています。でも安心してください。このクラスすべてをいじる必要はありません。順に簡単に説明していきましょう。

●CAboutDlgクラス

いわゆる「～について」で開くアバウトダイアログです。MFCではダイアログのスーパークラスとしてCDialogクラスが定義されており，その派生クラスです。クラスそのものをいじる必要はあまりないでしょう。test.cppのなかで宣言・定義されています。

●CMainFrameクラス

いわゆるメインウィンドウのクラスです。このクラスも意外にあまりいじることはありません。MainFrm.hで宣言，MainFrm.cppで定義されています。

●CTestAppクラス

一番根っこにあるアプリケーションクラスです。アプリケーションが起動されると，まずこのクラスのインスタンスが生成されます。Class ViewのグローバルのなかにあったtheAppというのは，このCTestAppクラスのインスタンスです。画面に直接見えるものではありませんが，アプリケーションの初期化の処理などを追加する場合があります。test.hで宣言，test.cppで定義されています。

●CTestDocクラス

ドキュメントクラス，つまり画面に表示する内容を保持するクラスです。たとえばエディタのようなアプリケーションの場合，読み込まれたファイルおよびその内容を管理するようなクラスです。アプリケーションを作る場合，主にこのクラスと次のビュークラスにメソッドを追加していくことになります。testDoc.hで宣言，testDoc.cppで管理されています。

●CTestViewクラス

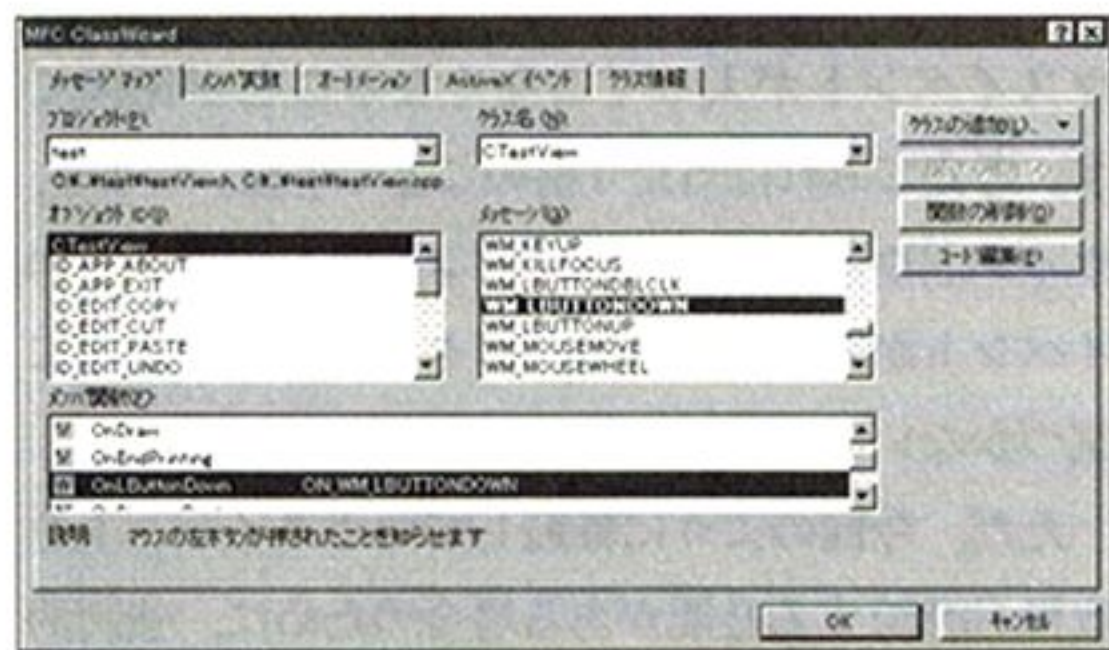
ビュークラス，つまりドキュメントを表示するビューポートを管理するクラスで，CMainFrameクラスのクライアントエリアにはまっています。testView.hで宣言，testView.cppで定義されています。

これらのクラス名やファイル名は，たとえばfooというプロジェクトを作った場合，デフォルトではCFooViewやfooView.cppなどといった名前になりますが，MFC AppWizardのステップ6で変更することもできます。

さて，では唐突にもプログラムのインプリメントを始めてみましょう。ちょっと不安だという人は，ソースにコメントがついていますので，ざっと目を通しておいてもいいでしょう。まずマウスカーソルがクライアント領域で左クリックされたときのイベントを取ってみます。クライアント領域に関することはCTestViewクラスですね。

しかし，いきなりtestView.cppファイルに手をつける必要はありません。まず，［表示］メニューの［ClassWizard］を選択してください。すると，MFC ClassWizardというダイアログが表示され，メッセージマップというタブが開いているはずです。ここで［クラス名］をCTestViewに，［オブジェクトID］もCTestViewを選択し，［メッセージ］のリストをずーっと下にスクロールさせてWM_LBUTTONDOWNを探してください。これが，CTestView上で左ボタンを押したときにくるメッセージです。そこでWM_LBUTTONDOWNをダブルクリックすると，下の［メンバ関数］にOnLButtonDown()が追加されるはずです（図10）。あとはそのOnLButtonDown()をダブルクリックしてやれば，自動的にメンバに登録され，ソースファイルが開いてそのメソッドの定義部分へカーソルがジャンプします。つまりそこのソースコードを書いてやれば，ウィンドウのクライアントエリアをクリックしたときに実

行されるようになるのです。なんて簡単なんでしょう。

図10　メッセージに対応する処理も簡単

さて，そのOnLButtonDown( )メソッドですが，2つの引数を取ります。

OnLButtonDown(UINT nFlags, CPoint point);

このメソッドをヘルプで探すとわかりますが，ひとつ目の引数はCtrlキーやマウスの右ボタンが同時に押されていた場合にセットされるフラグです(UINTというのはunsigned intをtypedefしたものです)。2つ目はマウスのボタンが押された座標を示し，CPointというクラスのインスタンスが渡されます。といってもそんなに構える必要はありません。とりあえず座標がほしいだけならば，point.xとpoint.yでそれぞれX座標とY座標を得ることができます。ここで座標についてちょっと注意しておきましょう。座標には画面左上を原点とし，右向きをX，下向きをYとしたスクリーン座標と，クライアント領域の左上を原点としたクライアント座標があります。相互の変換は難しくはありませんが，ここで拾った座標はもちろんクライアント座標ですので，クライアントに対してなにか行う場合には，その座標をそのまま使うことができます。

では，この座標を利用して，クライアント領域に点を打ってみましょう。描画を行うには，まずデバイスコンテキストという，描画を司るオブジェクトを取得しなければなりません。MFCを使っている場合は，デバイスコンテキストはCDCというクラスで扱うことができ，CWndクラス(ウィンドウの元となるクラスで，CTestViewクラスもこのクラスの派生クラスである)のGetDC( )メソッドでそのインスタンスを取得できます。点を描画するには，そのCDCのメソッドSetPixel( )を使いましょう。SetPixel( )は座標の取り方で多重定義されていますが，OnLButtonDown( )が座標をCPointクラスで渡されているので，

COLORREF SetPixel(POINT point,COLORREF crColor);

を使うことにします。CPointクラスはPOINT構造体から派生したものなので，そのまま渡すことができるのです。第2引数のCOLORREFというのは構造体で，RGB(red, green, blue)というマクロで作ることができます。ここでは赤い点にでもしておきましょう。これで点は打ったので終了なのですが，最後にデバイスコンテキストは使い終わったらReleaseDC( )メソッドで必ず解放しておかなければなりません。

ということで，OnLButtonDown( )メソッドはリスト1のようになります。この中で，説明しなかったTRACE( )という関数が出てきています。これは，MFCのデバッグ支援用のマクロで，アウトプットウィンドウに文字列を表示でき，printfと同様の可変数の引数を取ることができます。これはデバッグモードでのみ有効で，リリースモードでビルドすれば無視されます。ここでは，念のため，引数で渡された座標を表示するようにしてみました。

これでオーバーライドは完了しましたので，スーパークラスのCView::OnLButtonDown( )メソッドはコメントアウトしておきましょう。では，さっそくF7でビルドして，今度はF5(デバッグ実行)で実行してみてください。クライアント領域でマウスをクリックすると，赤い点が描画されるはずです(図11)。ただ，実行するとDeveloper Studioのアウトプットウィンドウが隠れて，代わりにデバッグウィンドウが表示されますので，[表示]メニューの[アウトプットウィンドウ]を選択して，アウトプットウィンドウも表示して，デバッグタブを選択してください。すると，test.exeでマウスをクリックして点を打つたびに，アウトプットウィンドウに座標が表示されるのがわかるでしょう。このように，TRACE( )マクロはちょっと変数を確認したい場合などに有効ですので，覚えておくと便利です。

とりあえず「動くアプリケーション」を作ってみたわけですが，いかがですか？　え？　ちょっと味気ない？　そうですね。点が打てるだけではあまりうれしくはないかもしれませんね。では，今度はこれを拡張して，線を引けるようにしましょう。流れとしては，

1)　左ボタンが押されたら座標を記録
　　↓
2)　マウスが動いたら以前の座標から今の座標へ線を引いて，新しい座標を記録
　　↓
3)　左ボタンが離されたら1へ戻る

といったところでしょうか。線を引くには，CDCクラスのMoveTo( )というメソッドと，LineTo( )というメソッドを使います。MoveTo( )で始点を設定し，LineTo( )で始点からそこまで直線を引くことができます。それには座標を記録しておくための変数が必要ですが，もうひとつ必要な変数があります。「現在ボタンが押されているかどうか」を示すフラグです。そうしないと，ボタンを押していないのに，マウスをただ動かしただけで勝手に線が引かれてしまいます。これは，1でセットして，3でリセットしてやればよいでしょう。

では，まずそれらの変数から宣言してやりましょう。ワークスペースウィンドウのFileViewからtestView.hをダブルクリックして開いてください。ここのCTestViewクラスのなかに次の2行を追加してください。

```
POINT m_OldPoint;
BOOL m_bActive;
```

これらの変数はクラス内部からしか参照することはないので，protectedの部分に記述してください。BOOLというのは真偽を表すタイプで，TRUE(真)とFALSE(偽)の値を取ることができます。といって，その実はint型で，以下のような記述がどこかのヘッダファイルにあると思って構いません。

```
typedef      int   BOOL;
#define FALSE  0
#define TRUE   1
```

忘れてはならないのは，この変数の初期化です。これはtestView.cppのなかにあるCTestViewクラスのコンストラクタ内で行えばいいでしょう。以下のように追加してください。

```
CTestView::CTestView()
{
// TODO: この場所に構築用のコードを追加してください。
    m_bActive = FALSE;
}
```

これでフラグはOKです。m_bOldPointのほう

リスト1　デバッグ用関数

```
void CTestView::OnLButtonDown(UINT nFlags, CPoint point)
{
            // TODO: この位置にメッセージ ハンドラ用のコードを追加する
かまたはデフォルトの処理を呼び出してください
            TRACE( "X:%d Y:%d¥n", point.x, point.y );
            CDC *pDC = GetDC();
            pDC->CDCSetPixel( point, RGB(255,0,0) );
            ReleaseDC( pDC );
//          CView::OnLButtonDown(nFlags, point);
}
```

図11　とりあえず実行……

は，左ボタンが押されたときに初期化されますので，ここでの初期化は不要です。

では先ほどと同じようにして，ClassWizardからボタンが離されたときのイベントとマウスが動いたときのイベントを取りましょう。左ボタンが離されたときのメッセージはWM_LBUTTONUP，動いたときのメッセージはWM_MOUSEMOVEですので，それぞれに対応するメソッドを作ってください。

あとはこれらのメソッドに適当にMoveTo()メソッドとLineTo()メソッドを追加してやるだけですが，ちょっと待ってください。先ほどのSetPixel()メソッドには引数で色を指定できましたが，LineTo()メソッドにはありませんね。LineTo()メソッドは，「現在CDCが選択しているCPenオブジェクト」で描画が行われるのです。したがって，まずCPenオブジェクトを作りましょう。それほど難しいことではありません。CPenのコンストラクタに線のスタイル，幅，色をほうり込んでやればいいだけです。いまはスタイルを実線，幅は1，色は赤としましょう。

```
CPen *pPen = new CPen(PS_SOLID,
1, RGB(255,0,0));
```

としてペンを作ったあと，

```
CPen *pOld = pDC->SelectObject
(pPen);
```

として選択します。戻り値は以前に選択されていたペンが返ります。使い終わったあとは，

```
pDC->SelectObject(pOld);
```

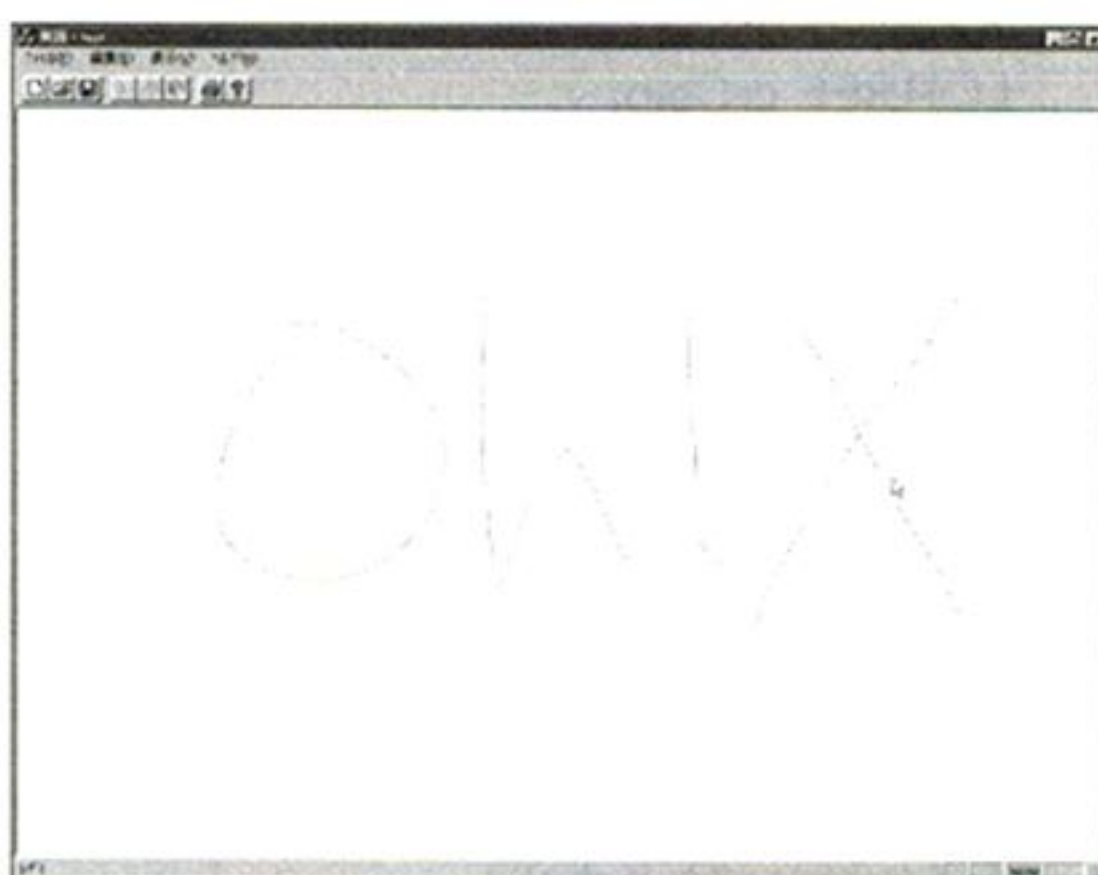
として念のために元のペンに戻し，

```
delete pPen;
```

としてペンを削除すればいいでしょう。

このようにして作ったメソッドはリスト2のようになります。これを見て，「いちいちGetDC()でデバイスコンテキストを取得するのは面倒だ」という人もいるかもしれません。確かにCDCのメンバを作って，ボタンを押したときに取得，離したときに解放したほうが，効率的に見えます。

しかし，このデバイスコンテキストをここでガメこんでしまうと，ほかで使えなくなってしまうのです。「そうはいっても，ほかで使ってないじゃん」というのは甘い考えです。たとえば，ウィンドウを更新するだけでもこのデバイスコンテキストを必要とし，その「ウィンドウを更新しなさいね」イベントはいつくるかわからないのです(ほかのウィンドウが上に重なった場合とか，壁紙が変更された場合とか)。そのときにデバイスコンテキストがほかで使われていると，ウィンドウを更新することができません。したがって，デバイスコンテキストはできる限りこまめに解放しろ，というのが通説です。ただ，ペンに関してはメンバを作って保存しておいてもよいでしょう。これ以上くどくど説明はしませんので，リストと実際の動作を見て各自で理解してください(図12)。

図12

さて，このアプリケーションをいじっていて，すでにおかしな点に気がついた人もいるでしょう。

1)　ほかのウィンドウが重なったりしたあと，再びtest.exeを最前面に持ってくると，以前に描いた線が消えている。

2)　ドラッグして線を描きながらウィンドウの外までマウスを動かし，そこでボタンを離すと，そのあとウィンドウ内にマウスを移動させただけでボタンを押していなくても線が描かれる。

1)については，デバイスコンテキストに線を出力しただけで，特に出力内容を記録していないために起こる問題です。デバイスコンテキストはあくまでもデバイスへのインタフェイスであって，バッファのようなものは普通持っていません。

したがって，デバイスコンテキストで出力した線は，ビデオメモリに保存されるだけで，そのビデオメモリがほかのウィンドウなどによって上書きされてしまうと，なにも残らないのです。これを回避するには，デバイスコンテキストに出力した内容を，アプリケーション側で記録しておく必要があります。こちらは大掛かりな修正が必要ですので，まず2)を片づけてしまいましょう。2)はマウスがクライアントエリアを出てしまうと，マウスイベントが入ってこなくなることが原因です。考えてみれば当たり前のことですね。つまり，ウィンドウの外でマウスボタンを離しても，そのイベントが渡らないので，ボタンを離したことに気づかないのです。

リスト2

```
void CTestView::OnLButtonDown(UINT nFlags, CPoint point)
{
		// TODO: この位置にメッセージ ハンドラ用のコードを追加する
かまたはデフォルトの処理を呼び出してください
		TRACE(  "OnLButtonDown:X:%d  Y:%d\n",  point.x,
point.y );
		m_bActive = TRUE;
		m_OldPoint = point;
//		CView::OnLButtonDown(nFlags, point);
}

void CTestView::OnLButtonUp(UINT nFlags, CPoint point)
{
		// TODO: この位置にメッセージ ハンドラ用のコードを追加する
かまたはデフォルトの処理を呼び出してください
		TRACE(  "OnLButtonUp:X:%d  Y:%d\n",  point.x,
point.y );
		m_bActive = FALSE;
//		CView::OnLButtonUp(nFlags, point);
}

void CTestView::OnMouseMove(UINT nFlags, CPoint point)
{
		// TODO: この位置にメッセージ ハンドラ用のコードを追加する
かまたはデフォルトの処理を呼び出してください
		if( m_bActive ){
				TRACE( "OnMouseMove:X:%d Y:%d\n",
point.x, point.y );
				CDC *pDC = GetDC();
				CPen *pPen = new CPen( PS_SOLID,
1, RGB(255,0,0) );
				CPen *pOld = pDC->SelectObject(
pPen );
				pDC->MoveTo( m_OldPoint );
				pDC->LineTo( point );
				pDC->SelectObject( pOld );
				delete pPen;
				ReleaseDC( pDC );
				m_OldPoint = point;
		}
//		CView::OnMouseMove(nFlags, point);
}
```

ただ，今回のように継続してマウスイベントを取得(追跡)する必要がある場合のために，強制的にマウスのイベントをぶんどる関数が用意されています。CWndクラスのSetCapture()がそれです。このメソッドを呼ぶと，ReleaseCapture()を呼ぶまでずっとマウスのイベントはそのウィンドウに渡されるようになります。ただし，このReleaseCapture()はCWndのメソッドではなく，Win32のライブラリ関数だったりします。ちょっと理不尽に感じるかもしれませんが，その辺は大人になればおいおいわかってくるでしょう(引数が必要ないからってだけの話ですが)。というわけで，OnLButtonDown()メソッドのなかに，

```
SetCapture();
```

を，OnLButtonUp()メソッドに，

```
::ReleaseCapture();
```

を追加するだけで問題解決です。ここで，ReleaseCapture()の前にコロンを2つ入れて，この関数がグローバルな関数である(クラスのメソッドではない)ということを明示しています。仮にCWnd(あるいはCView)のなかにReleaseCapture()というメソッドがあった場合，スコープによってそちらが優先されてしまわないようにということと，あとでソースを見返したときに，グローバルな関数を呼んでいるということが一目でわかるようにするためです。

■点の保存

「なんだ，Windowsのプログラムったって簡単じゃん」といい気になっていたあなた，ここからがヘビーになってくるんですよ。さて，さっきもいったように出力内容を保存しなくてはならないわけで，こういったデバイスコンテキストへの操作の保存用にメタファイルという形式があるのですが，こいつの操作にはちょっとクセがあるので，ここでは使うのをやめて，適当にフォーマットをでっち上げてしまいましょう。といっても，要はポイントを保存していけばいいだけですので，たいしたものではありません。ということで，次のような構造体を考えました。

```
typedef enum {
	PT_START,
	PT_LINK,
```

```
} POINTTYPE;
typedef struct {
        POINTTYPE type;
        POINT point;
} POLYLINE;
```

typeがPT_STARTの場合は始点，PT_LINKの場合は前の点との間に直線を引きます。この構造体の配列を用意しておき，クライアントエリアに線を引くと同時にこの構造体にも座標を格納していきます。では，この構造体はどこで宣言しましょうか。いままではビュークラスだけに変更を加えてきましたが，これはデータの内容そのものに関する事柄なので，初登場，ドキュメントクラス以外のなにものでもありません。ということですので，testDoc.hで上の構造体を宣言し，CTestDocクラスのメンバとしてその構造体の変数を登録しましょう。ただ，大きな配列を宣言するのは好ましくありませんので，ここはポインタにしておいて，コンストラクタで確保，デストラクタで解放するようにします。

この構造体へのポインタはビュークラスからも参照したいので，publicにしておいていいでしょう。コンストラクタでは，とりあえず構造体65536個分のメモリを割り当てておきます。決め打ちですが，これくらいあれば足りるでしょう。気に入らないという人は，各自で改良を加えてください。

さて，もうひとつ大切な変数が必要です。この構造体のどこまですでにデータが入っているかを示すインデックスです。これもint型のm_indexという名前にでもして，publicでドキュメントクラスに登録し，こちらはOnNewDocument()というメソッド内で初期化しておいてください。これは[ファイル]メニューの[新規作成]で呼ばれるメソッドですが，ドキュメントクラスが作成されたあとにも最初にも呼ばれます。

これで準備はできました。それではビュークラスから，マウスが操作されるたびにこの構造体に座標を格納していくことにしましょう。ビュークラスからドキュメントクラスのインスタンスを取得するには，CViewのGetDocument()を使います。こうして取得したインスタンスからm_pPolyLineとm_indexにアクセスして，座標を格納します。OnLButtonDown()メソッドならば次のような感じです。

```
CTestDoc *pDoc = (CTest
Doc *)GetDocument();
pDoc->m_pPolyLine[pDoc-
>m_index].type = PT_START;
pDoc->m_pPolyLine[pDoc-
>m_index].point = point;
pDoc->m_index++;
```

OnLButtonMove()では，PT_STARTがPT_LINKになるだけということでいいでしょう。これで格納もできました。あとはウィンドウの更新時に，このデータを元にクライアントエリアに書き込んでやればいいだけです。

ウィンドウの更新メッセージがきたときに呼ばれるメソッドはCTestViewのOnDraw()で，これは必ず必要になるものですので，ClassWizardで作らなくても，スケルトンで最初から作られています。このメソッドは引数としてCDCへのポインタが渡されていますので，必ずこのデバイスコンテキストを使って描画するようにしてください。というのは，このデバイスコンテキストには，すでに更新すべき領域のクリッピングや，その他の必要な属性が設定されているからです。もう少しいえば，このメソッドはプリンタへの印刷時にも呼び出され，その場合にはこのデバイスコンテキストはディスプレイデバイスコンテキストではなく，プリンタデバイスコンテキストなのです。つまり，このデバイスコンテキストに対して出力する限り，プログラマが特に意識する必要なく，そういった違いを吸収してくれるのです。

ここまで，ビュークラスに対して変更のあったメソッドだけを抜粋して，リスト3に示します。ついでですので，ペンをメンバに追加し，コンストラクタで作成，デストラクタで廃棄するように修正しておきました。これを見ると，OnDraw()メソッドでタイプがPT_LINKの場合，マウスイベントではわざわざ以前の座標を保存しておいてMoveTo()メソッドを使ったのに，それが省略されてLineTo()だけで簡単に済んでしまっていることがわかるかと思います。実はLineTo()メソッドはラインを引いたあとにその終点を始点として設定するという機能があるのです。ではなぜマウスイベントでそれを使わなかったというと，そこでは1回ごとにデバイスコンテキストを解放しており，それにより始点が初期化されてしまうのです。それに対し，OnDraw()メソッドではデバイスコンテキストを解放することなくループさせて描画を行っているので，始点の設定を省略できるのです。同様の理由で，ペンをデバイスコンテキストに1回1回設定するのも省いています。

リスト3

```
CTestView::CTestView()
{
        // TODO: この場所に構築用のコードを追加してください。
        m_pPen = new CPen( PS_SOLID, 1, RGB(255,0,0) );
}

CTestView::~CTestView()
{
        delete m_pPen;
}

void CTestView::OnDraw(CDC* pDC)
{
        CTestDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        CPen *pOld = pDC->SelectObject( m_pPen );
        for( int i=0; i<pDoc->m_index; i++ ){
                switch( pDoc->m_pPolyLine[i].type ){
                case PT_START:
                 pDC->MoveTo( pDoc->m_pPolyLine[i].point );
                 break;
                case PT_LINK:
                 pDC->LineTo( pDoc->m_pPolyLine[i].point );
                 break;
                default:
                 break;
                }
        }
        pDC->SelectObject( pOld );
        // TODO: この場所にネイティブ データ用の描画コードを追加します。
}

void CTestView::OnLButtonDown(UINT nFlags, CPoint point)
{
        // TODO: この位置にメッセージ ハンドラ用のコードを追加するかまたはデフォルトの処理を
                呼び出してください
        TRACE( "OnLButtonDown:X:%d Y:%d¥n", point.x, point.y );
        m_bActive = TRUE;
        m_OldPoint = point;
        SetCapture();
        CTestDoc *pDoc = (CTestDoc *)GetDocument();
        pDoc->m_pPolyLine[pDoc->m_index].type = PT_START;
        pDoc->m_pPolyLine[pDoc->m_index].point = point;
        pDoc->m_index++;
        pDoc->SetModifiedFlag();
//      CView::OnLButtonDown(nFlags, point);
}

void CTestView::OnLButtonUp(UINT nFlags, CPoint point)
{
        // TODO: この位置にメッセージ ハンドラ用のコードを追加するかまたはデフォルトの処理を
           呼び出してください
        TRACE( "OnLButtonUp:X:%d Y:%d¥n", point.x, point.y );
        m_bActive = FALSE;
        ::ReleaseCapture();
//      CView::OnLButtonUp(nFlags, point);
}

void CTestView::OnMouseMove(UINT nFlags, CPoint point)
{
        // TODO: この位置にメッセージ ハンドラ用のコードを追加するかまたはデフォルトの処理を
           呼び出してください
        if( m_bActive ){
                TRACE( "OnMouseMove:X:%d Y:%d¥n", point.x, point.y );
                CDC *pDC = GetDC();
                CPen *pOld = pDC->SelectObject( m_pPen );
                pDC->MoveTo( m_OldPoint );
                pDC->LineTo( point );
                pDC->SelectObject( pOld );
                ReleaseDC( pDC );
                m_OldPoint = point;
                CTestDoc *pDoc = (CTestDoc *)GetDocument();
                pDoc->m_pPolyLine[pDoc->m_index].type = PT_LINK;
                pDoc->m_pPolyLine[pDoc->m_index].point = point;
                pDoc->m_index++;
                pDoc->SetModifiedFlag();
        }
//      CView::OnMouseMove(nFlags, point);
}
```

リスト4

```
BOOL CTestDoc::OnNewDocument()
{
        if (!CDocument::OnNewDocument())
                return FALSE;

        // TODO: この位置に再初期化処理を追加してください。
        // (SDI ドキュメントはこのドキュメントを再利用します。)
        m_index = 0;
        SetModifiedFlag( FALSE );
        return TRUE;
}

BOOL CTestDoc::OnOpenDocument(LPCTSTR lpszPathName)
{
        if (!CDocument::OnOpenDocument(lpszPathName))
                return FALSE;

        // TODO: この位置に固有の作成用コードを追加してください
        FILE *fp  = fopen( lpszPathName, "rb" );
        if( fp ){
                fread( &m_index, sizeof(int), 1,
                fp );
                fread( m_pPolyLine, sizeof
                (POLYLINE), m_index, fp );
                fclose( fp );
                SetModifiedFlag( FALSE );
        }
        return TRUE;
}

BOOL CTestDoc::OnSaveDocument(LPCTSTR lpszPathName)
{
        // TODO: この位置に固有の処理を追加するか、または基本クラス
                を呼び出してください
        FILE *fp  = fopen( lpszPathName, "wb" );
        if( fp ){
                fwrite( &m_index, sizeof(int), 1,
                fp );
                fwrite( m_pPolyLine, sizeof
                (POLYLINE), m_index, fp );
                fclose( fp );
                SetModifiedFlag( FALSE );
                return TRUE;
        }
        return FALSE;
//      return CDocument::OnSaveDocument(lpszPathName);
}
```

■データのロード/セーブ

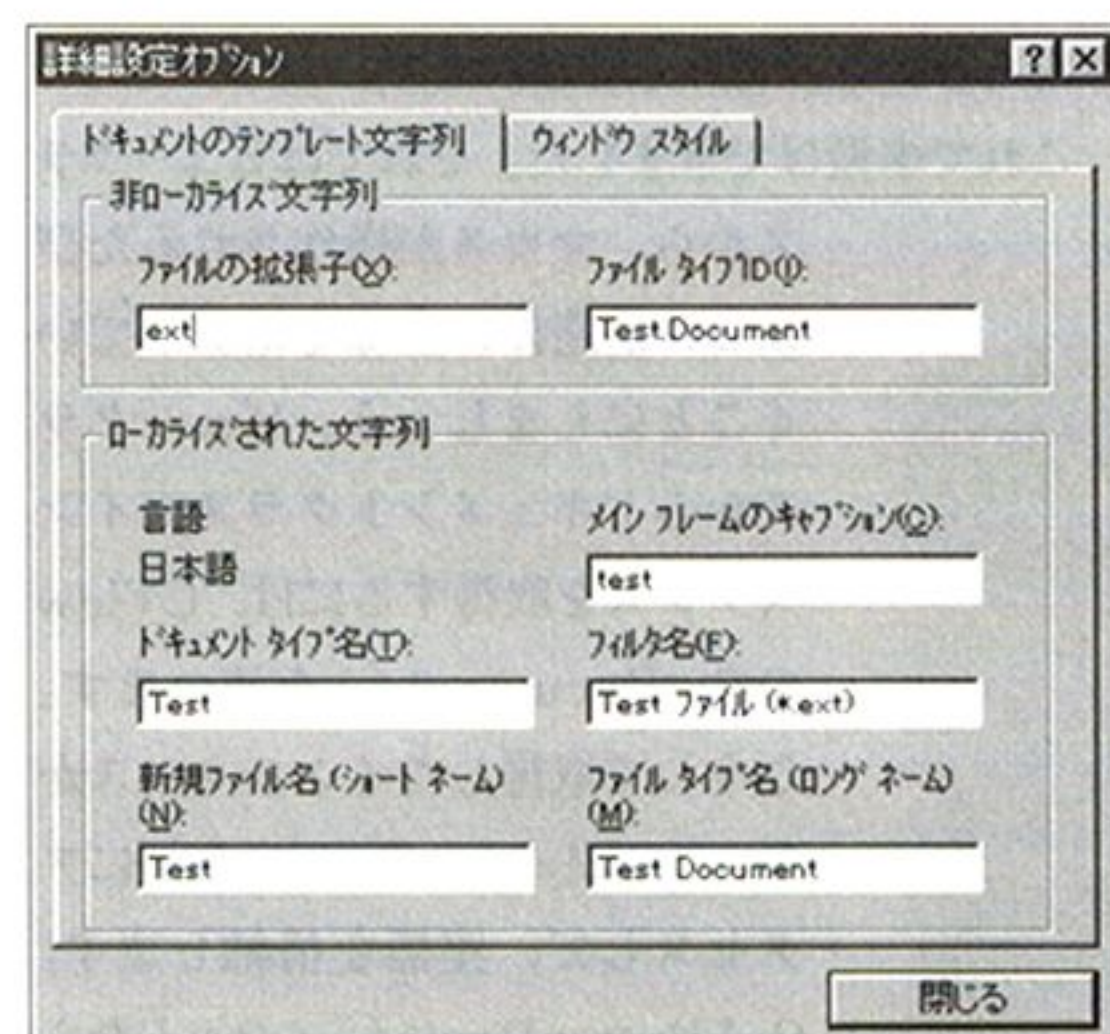
さて，ここまできたのですから，一気にデータのロード/セーブもやってしまいましょう。これにはClassWizardでCTestDocクラスにOnOpenDocument()およびOnSaveDocument()を追加し，そこにロードとセーブのコードを記述します。引数にはファイル名が渡されますので，そのファイルを開いて読み出し，あるいは書き込みを行うだけです。フォーマットは，インデックスを頭の4バイトに置き，あとは例の配列をバイナリで書き込めばいいでしょう。

ここでもうひとつ説明しておきたいことがあります。ModifiedFlagについてです。これはドキュメントに変更が加えられたかを示すフラグで，ドキュメントクラスが内部に保持するフラグです。これをTRUEにしておくとドキュメントに変更が加えられたことを示し，そのままアプリケーションを終了しようとすると，「変更を保存しますか？」というメッセージを出して，保存を促してくれるのです。このフラグを設定するのはCTestDocクラスのSetModifiedFlag()で，先ほどは述べませんでしたが，リスト3ではすでにドキュメントを更新するときにはこのフラグをセットするように記述してあります。

ではこのフラグはいつリセットすればいいでしょうか。保存に成功したときと，あとはロードに成功したとき，新規作成したときにもリセットしておいていいでしょう。このことも踏まえて，ドキュメントクラスに変更を加えたメソッドの抜粋をリスト4に示します。簡単に説明を済ませましたが，ソースをじっくり読み下してモノにしてください。

実行すると，どうですか？　ほかのウィンドウに隠れたりしても，そのあとにちゃんと表示(というか書き直し)しますね？　しかもOnDraw()をインプリメントしたことで，描いたものを印刷することもできるようになっています。適当にさらさらと落書きして，楽しんでください。ただし，保存できるポイントの個数は65536個にしましたが，オーバーフローのチェックはまったくしていませんので，ほどほどに。なにか描けたら保存もしてみましょう。[ファイル]メニューの[名前を付けて保存]を選択したら，ファイルダイアログで適当なファイル名をつけて保存してみてください。そのあとは[新規作成]でもしていったんクリアして，再び今保存したデータを開いてみれば，ちゃんとロード/セーブできていることがわかるでしょう。

ただちょっと気になることがありますね。[ファイルの種類]で，たとえばWordなら[Word文書（*.doc)]とか出てくるのに，これは[すべてのファイル(*.*)]しかありません。適当に名前をつけても，拡張子が補完されることもありません。こういったことは，ずいぶん前に戻りますが，実はプロジェクトを作成するときに設定できたのです。これは，MFC AppWizardのステップ4，[詳細設定]というボタンにこっそり(？)隠されています。ここで[ファイルの拡張子]と[フィルタ名]を適当に設定しておけば，ファイルダイアログで勝手に使ってくれたのです(図13)。今回は拡張子名を考えるのも面倒だったので無視しましたが，自分でなにか作るときには設定してみてください。もちろんあとからリソースのなかのString Tableをいじることで設定することもできるのですが，ここではその方法については触れません。

図13　ファイル種類の設定

■色の選択とダイアログ

ずいぶんアプリケーションらしくなってきましたが，もうひとつもの足りませんね。やっぱり赤い線だけではね……ということで，線の色を選択できるようにしてみましょう。

さて，初めてのユーザーインタフェイス部分のプログラミングです。こういうインタフェイスはダイアログが基本ですが，ダイアログには2種類あることをご存じでしょうか。ひとつは，そのダイアログが開いている間，その親(アプリケーションのウィンドウなど)の操作ができなくなるものです。ファイルダイアログなどもそうですし，普段目にするダイアログの多くはこちらです。なにかを選択しなければ先へ進めない処理の場合にこちらを使います。もうひとつは親であるウィンドウと同時に操作できるダイアログです。こちらは逐一設定を変更してアプリケーションを操作することができます。前者をモーダルダイアログ，後者をモードレスダイアログといいます。

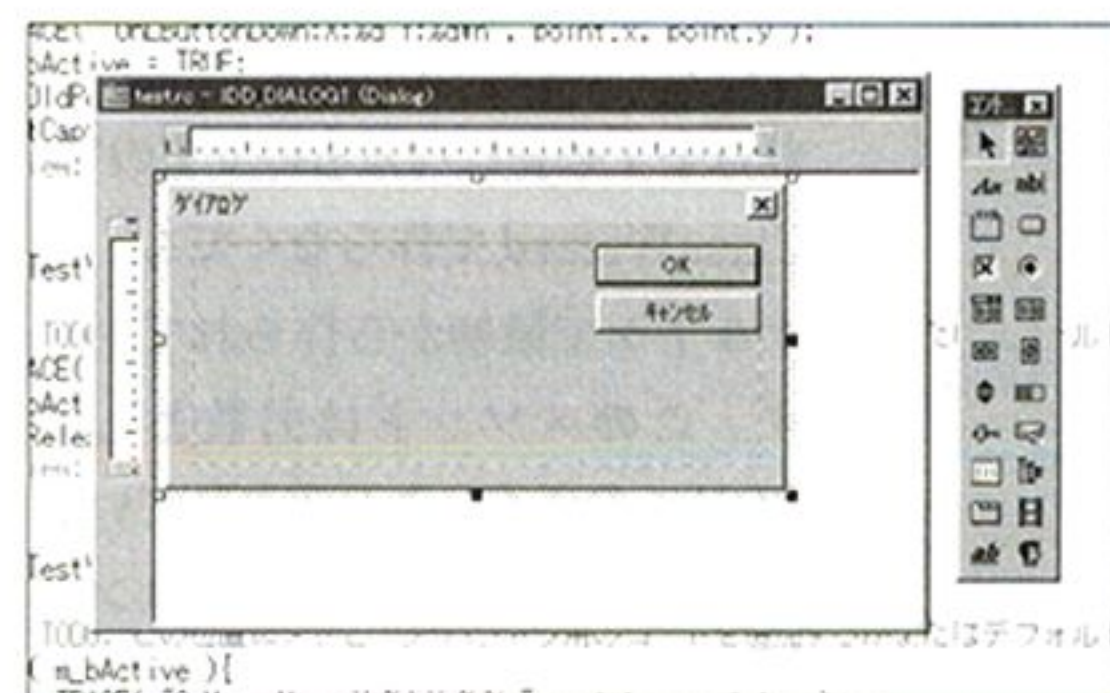



図14　ダイアログエディタ

いまの場合はモードレスが理想ですが，初めてのダイアログということで，簡単なモーダルにしてみましょう。ダイアログを作るには，まずテンプレートを作成します。[挿入]メニューの[リソース]を開き，そこから[Dialog]を選択して[新規作成]ボタンを押してください。すると，ワークスペースのResourceViewにIDD_DIALOG1というダイアログが追加され，画面にはOKとキャンセルボタンだけが載ったダイアログが表示されたウィンドウが(なんかややこしいな)が表示されるはずです。ついでにいろいろなアイコンが載ったツールバーも表示されたでしょう。これがダイアログエディタです(図14)。なんか，やっと"Visual"らしくなってきましたね。

とりあえずはそのダイアログのタイトル辺りで右クリックし，メニューから[プロパティ]を選択し，キャプションを「色の指定」にでも変更しましょう(図15-A)。すると，ダイアログ自体のタイトルバーの表示が「色の指定」に変わりましたね。ではダイアログにコントロールを載せていきましょう。コントロールツールバーの右の列の上から4番目のラジオボタンらしきアイコンを選択し，ダイアログの上で適当にドラッグしてみてください。すると，"Radio1"というキャプションのついたラジオボタンが作成されるはずです。ではさらにこのラジオボタンの上で右クリックし，プ

図15-A　ダイアログタイトルの変更

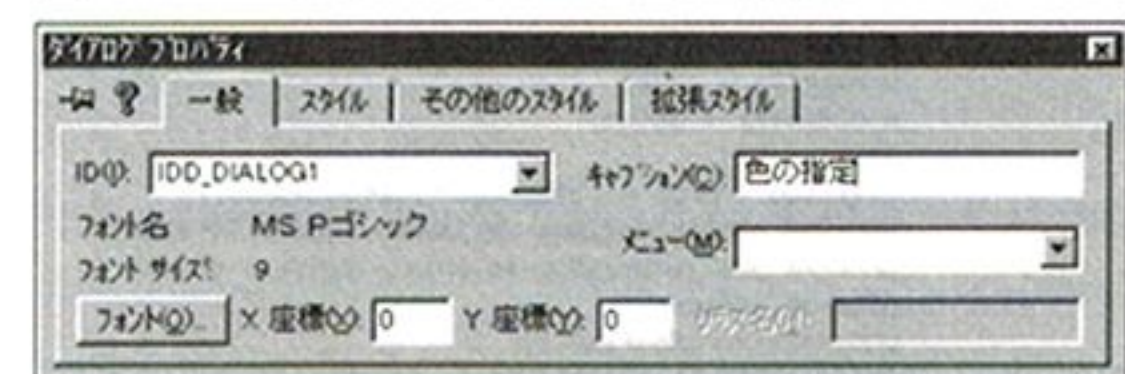



図15-B　ボタンの設定だとこんな感じ

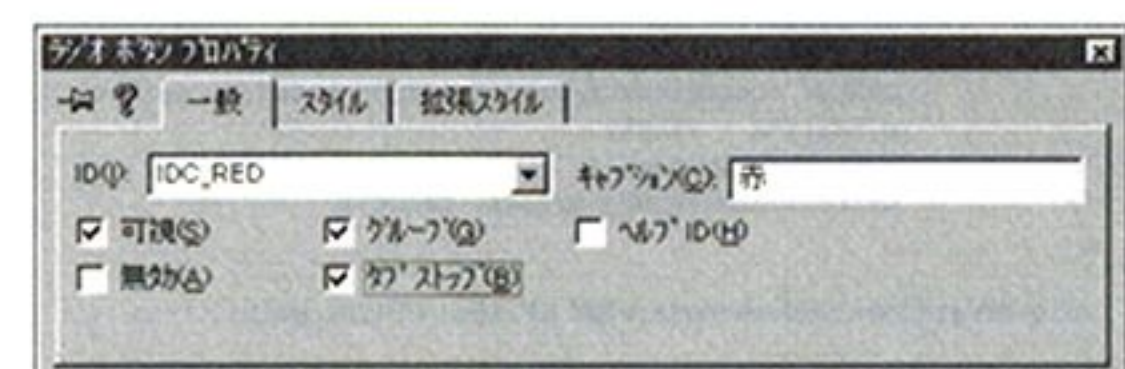
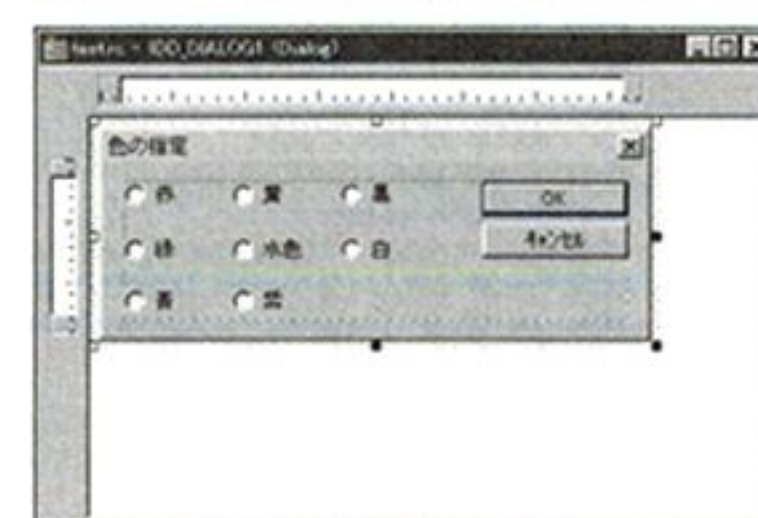
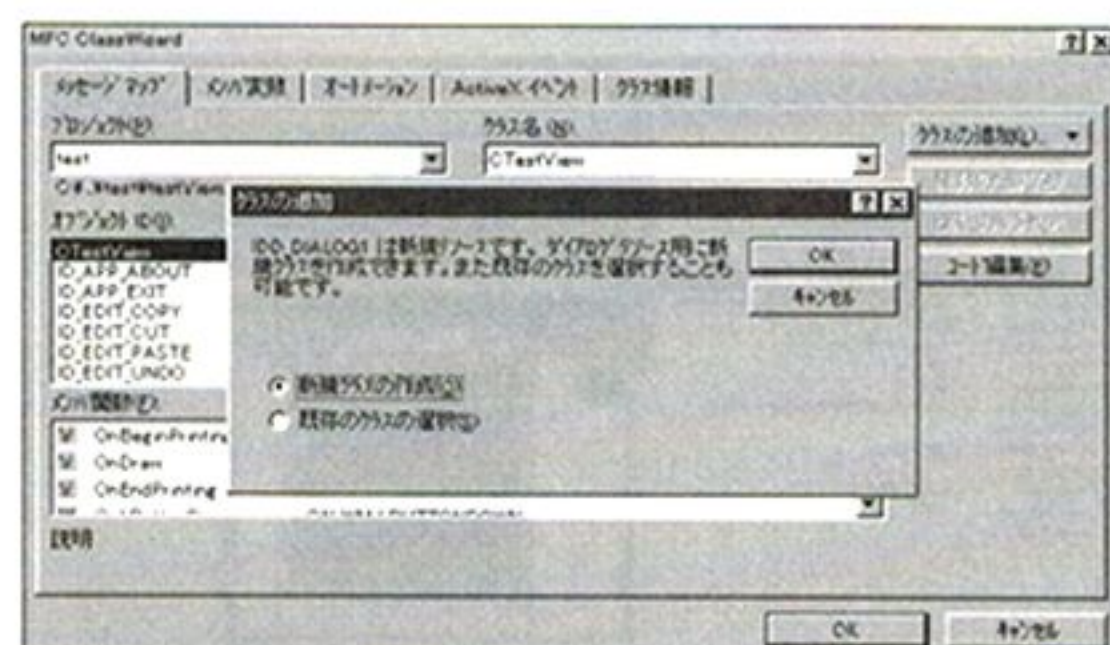



図16　どんどん作っていく

図17　クラスを追加する

図18　赤しかないけど気にしない

図20　タイプとメンバを確認

ロパティを表示してみてください。ここで，IDも適当に決められていますが，このままではわかりにくいので，IDC_REDと変更し，キャプションも「赤」にしましょう(図15-B)。あと，これだけは[グループ]と[タブストップ]もチェックしておいてください。

このようにして，ほかの色もコントロールを作っていくのですが(図16)，ほかは[グループ]と[タブストップ]をチェックする必要はありません。先頭の「赤」だけに［グループ］と［タブストップ］をチェックしたのは，次に[グループ]が設定されたラジオボタンが現れるまでを1グループとして(いまはほかにグループ設定したものがないので，すべてが1グループとなる)，そのグループ全体でタブストップを設定するという意味になります。

エディットができたら，[レイアウト]メニューの[テスト]で動作を見てみるとよいでしょう。最初にどれかを押すまでは，ラジオボタンがどれも選択されていない状態になりますが，これは気にする必要はありません。タブやカーソルキーを動かして，フォーカスの動き方も見ておいてください。

楽しかった(?)ビジュアルエディットもここまでです。今度はいま作ったダイアログのタイトルバーをダブルクリックすると，お馴染みのMFC ClassWizardが開き，[クラスの追加]というダイアログが表示されます(図17)。ここで[新規クラスの作成]を選択して，OKしてください。すると[クラスの新規作成]というダイアログが開きますのでクラス名を“CColDlg”とでもしましょう。

ここで，ダイアログIDにいま作成したテンプレートのID，IDD_DIALOG1が入っていることを確認しておいてください。するとMFC Class Wizardの[クラス名]のなかに“CColDlg”が増えますが，これはいまはほっておいて，［メンバ変数］タブを開いてください。ここで[クラス名]が“CColDlg”になっていることを確認したら，その下のリストボックスを見てください(図18)。あれ？　いま作ったIDC_REDはありますが，その他のラジオボタンのIDがありませんね。まあとりあえず気にしないで，そのIDC_REDをダブルクリックしてみてください。今度は[メンバ変数の追加]です(図19)。いろんなダイアログが出てきていやになりそうですが，もう少しの辛抱です。

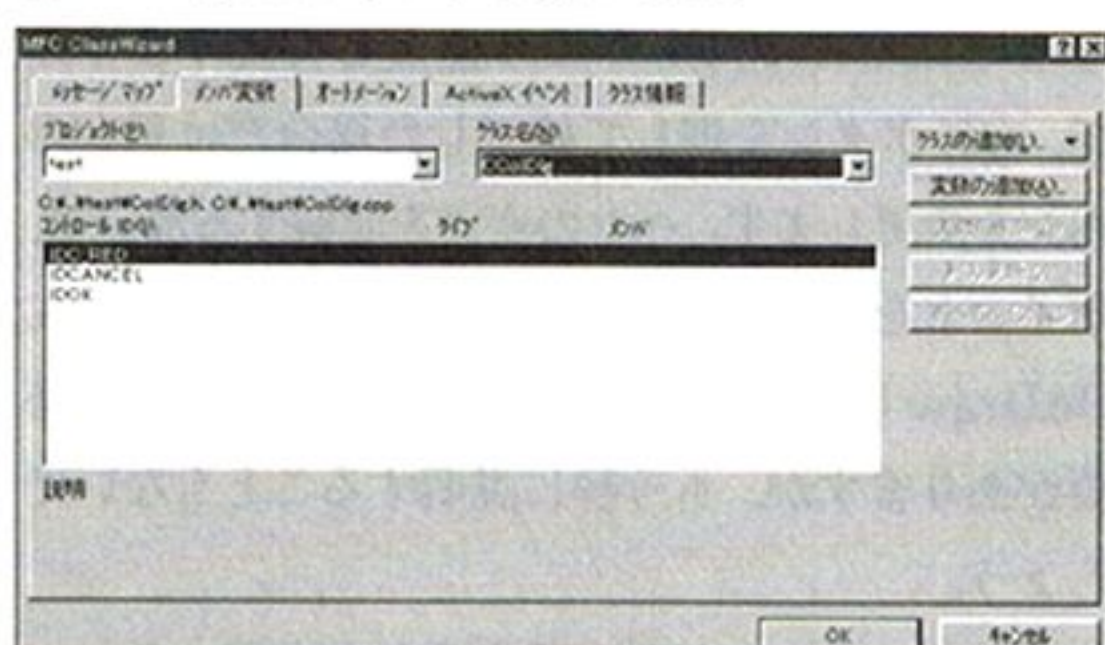
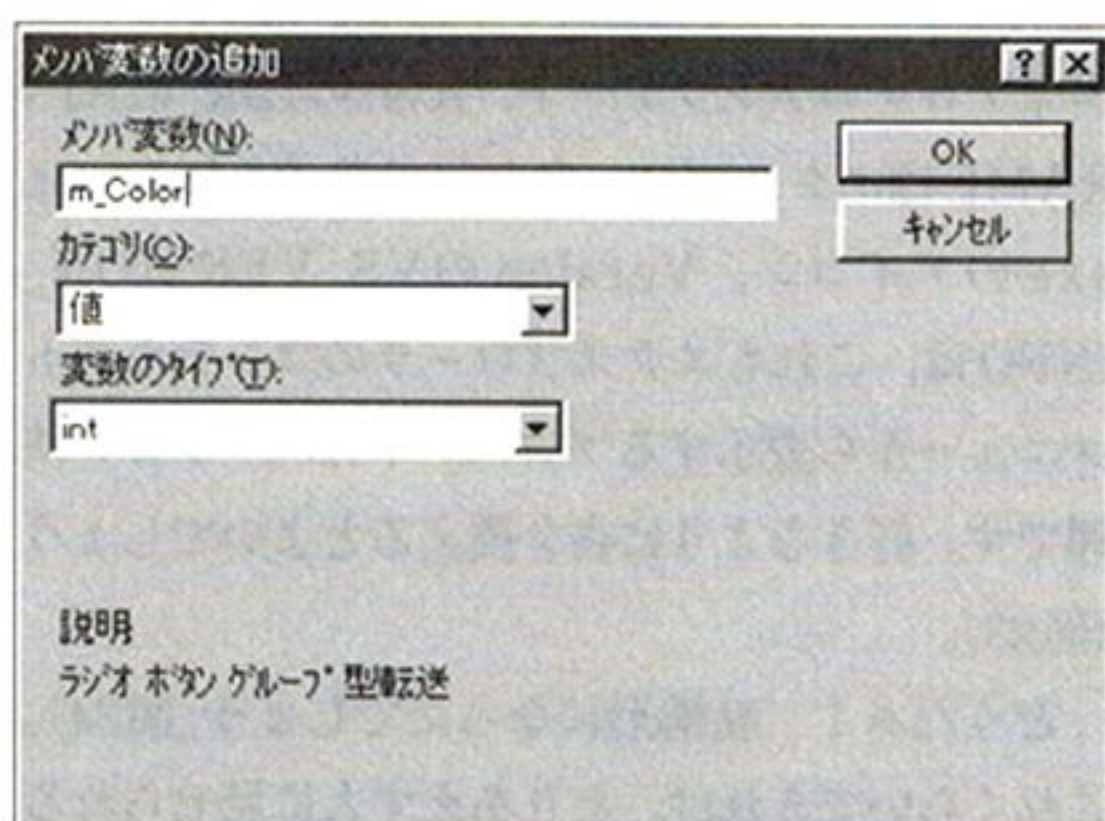



図19　メンバ変数の追加

ここでは，コントロールに変数を割り当てることができるのです。メンバ変数をm_Colorとでもしましょう。[カテゴリ]は「値」，[変数のタイプ]は「int」となっています。そしてその下，「ラジオボタングループ型転送」という説明があります。ここでピンと来たでしょうか。ラジオボタンは選択されているかどうかですから，ひとつだけならBOOLでいいはずですよね。つまり，グループ化されたラジオボタンは，変数ひとつで何番目が選択されているかを教えてくれるのです。どうせどれかひとつしか選択できないのがラジオボタンですから，1つひとつ押されているか調べるよりも，このほうがずっと便利です(これはMFCによって提供されるもので，Win32 SDKだけでは1つひとつ調べることになります)。

ではここはOKして，MFC ClassWizardでIDC_REDに変数が追加されていることを確認したら(図20)，これもOKしてください。これでダイアログのクラスの作成は終了です。ふ～，やれやれ，って安心しないでください。まだクラスができただけで，そのインスタンスを作るには至ってないんですよ。ではこのダイアログはなにをしたときに表示しましょうか。たいていは[色の指定]とかなんとかいうメニューを選択すると開くもんですよね。

ではメニューを追加しなくてはいけませんね。再びResourceViewです。ここのMenuのなかにすでにIDR_MAINFRAMEというメニューのスケルトンがありますので，それに追加してみましょう。こいつをダブルクリックして開いたら，[ヘルプ]のさらに右をダブルクリックしてみてください。階層状にするのも面倒なんで，ここにつけてしまいましょう。ポップアップのチェックをはずしたら，IDにはID_SELECTCOLOR，キャプションは「色の指定(&C)...」としましょう。ここで‘(&C)’というのはショートカットキーで，このように記述するだけで，Alt+Cと押したときにこのメニューが実行されるようになります。いちばん下のプロンプトは，このメニューにカーソルをあわせたときにステータスバーに表示される文字列ですので，これも適当に書いておきましょう(図21)。

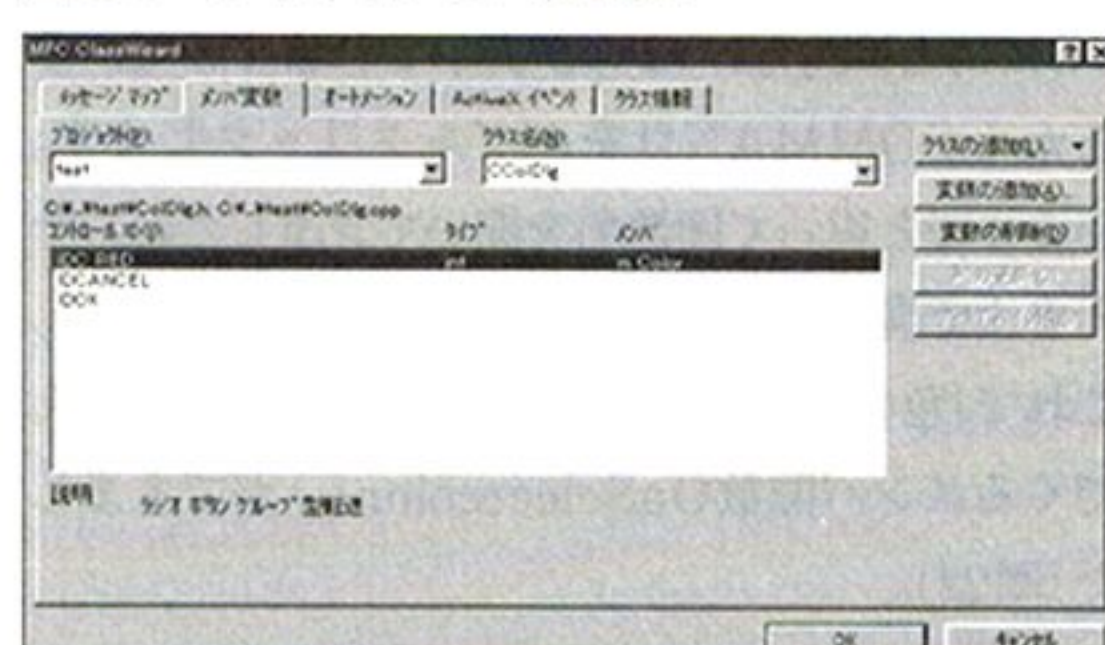
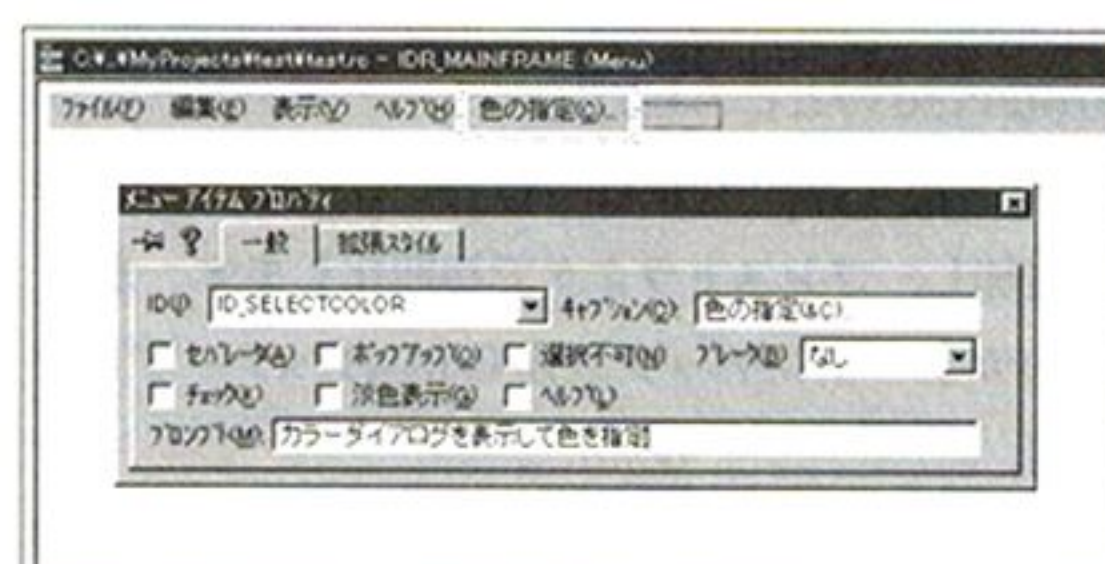
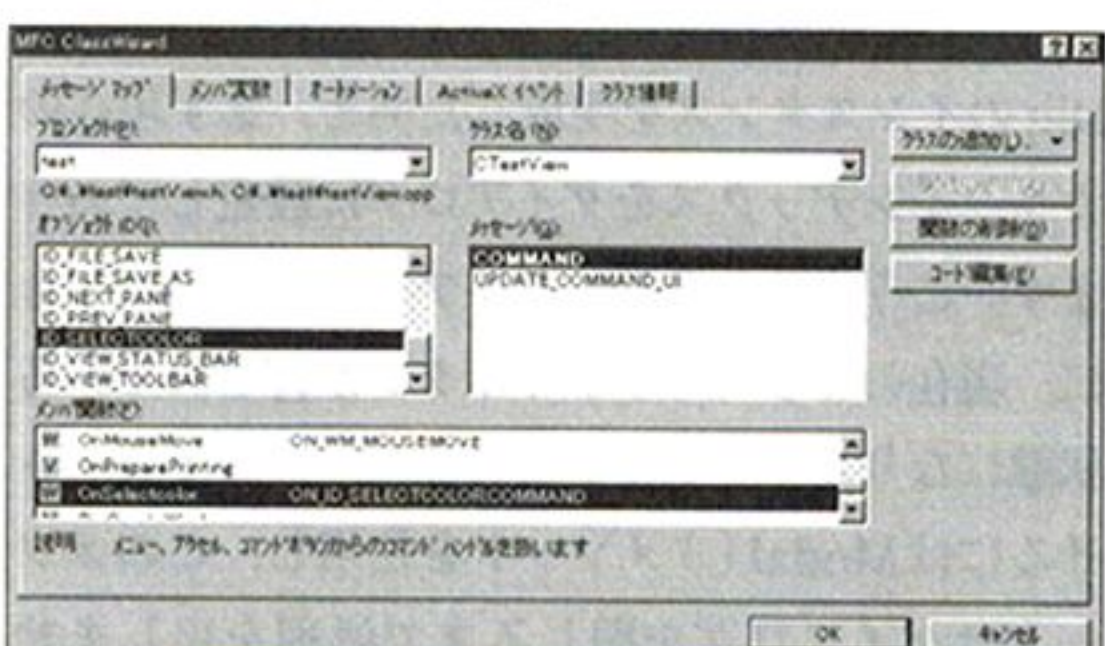



図21　ステータスバーへのコメントを指定

メニューでやることはこれだけです。あとは，このメニューが選択されたときに起こるイベントを受けるメソッドを作り，そこでダイアログを開き，選択された色でペンを作り直す・だ・けです。

さて，毎度お馴染みMFC ClassWizard。メッセージを受ける，つまりダイアログを開く，もっといえば色を管理するのはどこがいいでしょうか。ドキュメントクラスのほうが正しい気もしますが，実際に使うのはビュークラスですので，ビュークラスにしてしまいましょう。というわけで，クラス名にCTestViewを選択したら，今度はオブジェクトIDからいま作ったメニューのID，ID_SELECTCOLORを選択してください。すると，

図22　呼び出し側に登録

メッセージにはCOMMANDとUPDATE_COMMAND_UIの2つが表示されますので，そこからCOMMANDをダブルクリックします。いままでと違って関数名を聞いてきましたね。そのままで構わないので，OKしてしまいましょう。これで[色の指定]メニューを選択したときに飛んでくるメンバ関数OnSelectcolor( )ができました(図22)。

次はダイアログの表示ですが，その前にCTestViewクラスに現在のカラーインデックスを保持する変数を追加しておいてください。protectedのところに，

```
int m_ColorIndex;
```

を追加し，コンストラクタで初期化します。また，このカラーインデックスからRGBへテーブルとして，

```
static COLORREF Color[8] = {
        RGB( 255, 0, 0 ),
        RGB( 0, 255, 0 ),
        RGB( 0, 0, 255 ),
        RGB( 255, 255, 0 ),
        RGB( 0, 255, 255 ),
        RGB( 255, 0, 255 ),
        RGB( 0, 0, 0 ),
        RGB( 255, 255, 255 )
};
```

という配列を用意しておきましょう。また，このCTestViewでCColDlgが使えるように，testView.cppにColDlg.hをインクルードしておきましょう。さらに，ペンの色の変更もドキュメントのデータとして保存しなければなりませんので，データの構造体を以下のように拡張し，関連部分も修正します。

```
typedef enum {
        PT_START,
        PT_LINK,
        PT_COLOR,
} POINTTYPE;
typedef struct {
        POINTTYPE type;
        union {
                POINT point;
                COLORREF color;
        } var;
) POLYLINE;
```

ではOnSelectcolor( )です(リスト5)。CColorDlgのインスタンスを生成したあと，まず現在のカラーインデックスをダイアログに設定しておきます。こうすることで，ダイアログを開いたときに，現在のカラーのラジオボタンが押されている状態になります。ダイアログをモーダルで表示させるにはModal( )メソッドを使用し，このメソッドはダイアログを閉じるまで処理を戻しません。もしOKが押されればIDOKを，キャンセルならばIDCANCELが戻り値として返りますので，OKボタンを押して終了した場合だけ現在のペンを変更します。ペンの変更を記録する部分は，ポイントを記録する方法と同じですね。もちろんOnDraw( )メソッドもこれにあわせて修正の必要がありますが，もう特に説明することもないでしょう。

せっかくリソースまで手を出したので，ちょっと紹介しておきましょう。DialogのなかのIDD_ABOUTBOXは[バージョン情報]で表示されるダイアログのテンプレート，IconのIDR_MAINRAMEはエクスプローラなどで表示されるtest.exeのアイコン，VersionのVS_VERSION_INFOは，これもエクスプローラのコンテキストメニューから表示するプロパティのバージョン情報です。好きなように書き換えるとよいでしょう(図23)。

どうだぁ！　結構形になったでしょう(図24)。これくらいできれば，とりあえず人に見せられるレベルじゃないでしょうか。あとは線のスタイルとか太さなんかも設定できるといいかもしれませんが，その辺は自分で挑戦してください。リストボックスとかスピンボタンコントロールなんかの使い方も必要になりそうですが，それもご自分で。なに，ヘルプを探せばどっかに書いてありますって(そりゃそうだ)。要は解説どおりにやるだけじゃなくて，自分で調べてほしいってことです。そうこうやっているうちに新しい発見もあるかもしれないし。あと，アンドゥも簡単に実装できますよね。

リスト5　OnDrawの変更

```
void CTestView::OnDraw(CDC* pDC)
{
        CTestDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        CPen *pPen, *pOld;
        pPen = new CPen( PS_SOLID, 1, Color[0] );
        pOld = pDC->SelectObject( pPen );
        for( int i=0; i<pDoc->m_index; i++ ){
                        switch( pDoc->m_pPolyLine[i].type
){
                        case PT_START:
                         pDC->MoveTo( pDoc->m_pPolyLine
                         [i].var.point );
                         break;
                        case PT_LINK:
                         pDC->LineTo( pDoc->m_pPolyLine
                         [i].var.point );
                         break;
                        case PT_COLOR:
                         delete pPen;
                         pPen = new CPen( PS_SOLID, 1,
                         pDoc->m_pPolyLine[i].var.color );
                         pDC->SelectObject( pPen );
                         break;
                        default:
                         break;
                         }
        }
        pDC->SelectObject( pOld );
        delete pPen;
        // TODO: この場所にネイティブ データ用の描画コードを追加します。
}

void CTestView::OnSelectcolor()
{
        // TODO: この位置にコマンド ハンドラ用のコードを追加してください
        CColDlg dlg;
        dlg.m_Color = m_ColorIndex;
        if( dlg.DoModal()==IDOK ){
                        m_ColorIndex = dlg.m_Color;
                        delete m_pPen;
                        m_pPen = new CPen( PS_SOLID, 1,
Color[m_ColorIndex] );
                        CTestDoc *pDoc = (CTestDoc *)
                        GetDocument();
                        pDoc->m_pPolyLine[pDoc->m_index]
                        .type = PT_COLOR;
                        pDoc->m_pPolyLine[pDoc->m_index]
                        .var.color = Color[m_ColorIndex];
                        pDoc->m_index++;
                        pDoc->SetModifiedFlag();
        }
}
```

図23　アイコンを指定する

図24　これで一応完成

問題はアンドゥしたあとの画面の書き換えですが，白で塗りつぶして書き直すよりも，ウィンドウを更新させたほうがスマートです。ヒントはInvalidate( )メソッド(ってそのままだけど)。そうそう，いままでずっとデバッグモードでビルドしてたんですが，それではデバッグ用の不要なコードが山ほどくっついて実行ファイルが1MBを超えるうえに，実行速度が著しく低下しますので，配布するときなんかはリリースモードでビルドし直しましょう。それには，[ビルド]メニューの[アクティブな構成の設定]メニューを選択し，

test - Win32 Release

ってほうを選択してください。

わかってると思いますが，この辺りはまだ基礎の基礎です。本当はやらなきゃいけないことがまだまだあるのですが，これ以上ここでやってもついてきてくれる人がどれほどいるか疑問ですし。「困ったときのthisポインタ」とかいって，thisポインタ出てこなかったけど。ま，また次回余裕があれば(それ以前に本が出れば)，さらにこの先の解説なんかできるかもしれません。レジストリとかMDIとか，あと私の専門分野として画像なんかも扱いたかったんですけどね。もし少しでも期待してくれているのなら，立ち読みなんかしてないで，買って読者はがきを出してください。ここまで立ち読みでたどりつく人がいるとも思えないけど。

JavaScriptから始めるプログラミング

古籏一浩/Furuhata Kazuhiro

現在，もっとも手軽なプログラミング環境でもあるJavaScript。Web上でのニーズも高く，しかも，HTMLでサポートされているマルチメディア系のコンテンツが扱える。プログラミングの手始めにはいちばん向いているのかもしれない。

■JavaScriptってなんだ？

JavaとJavaScriptどっちも耳にしたことはあると思います。さて，この2つどう違うのでしょう?

「JavaとJavaScriptって同じじゃないの？」

残念，違います。名前は似ていますが，似て非なるものです。つまりまったく「別モノ」。Javaは，どの機種でもほとんど同じ動作をします。ただ，ちょっと玄人向けといった感じでしょうか。JavaScriptも，機種を問わずほぼ同じ動作をします。ただJavaと違って，複雑な部分が少なく，手軽にプログラムすることができます。JavaScriptのプログラムを作る場合，開発に必要なものはエディタだけです。もちろんJavaScriptを動かすためのWebブラウザも必要ですけど。

JavaScriptに対応しているブラウザは以下のとおりです。

Netscape Navigator 2.0以降
Internet Explorer 3.0以降
(Mac版は3.01以降*1)
Opera 3.0以降(現在はWin版のみ)

ここで対象としているブラウザはNetscape 4以降，Explorer 4以降です。またHTML(Hyper Text Markup Language)およびスタイルシート定義の基本的な知識はすでに習得しているものとして話を進めていきます。あとは，習うより慣れろ，実践あるのみです。

■プログラムはHTMLの中に書く！

JavaScriptでプログラムを作る場合，どうしても知っておかなければならないことがあります。それは，JavaScriptのプログラムはHTMLの文書中に書く，ということです。JavaScriptの基本形は以下のようになります。

```
<SCRIPT Language="JavaScript">
<!--
  プログラム
// -->
</SCRIPT>
```

これだけです。HTML文書の中であれば，どこに置いてもたいてい動きます。が，通常は<HEAD>～</HEAD>の間または<BODY>～</BODY>タグで挟まれた部分に入れます。つまり基本パターンとして以下のようになります。

```
<HTML>
<HEAD>
<TITLE>Sample</TITLE>
<SCRIPT Language="JavaScript">
<!--
  ここにプログラムを書く
// -->
</SCRIPT>
</HEAD>
<BODY>
  本文
<SCRIPT Language="JavaScript">
<!--
  ここにプログラムを書く
// -->
</SCRIPT>
</BODY>
</HTML>
```

これ以外にイベント指定部分にもJavaScriptの命令を書くことができます。これも基本形は以下のようになります。

```
<タグ名 イベント名=JavaScriptの命令>
例：
<A HREF="../" onMouseOver="alert('SHARP MZ-700')
```

■命令の基本パターン

JavaScriptの命令には基本パターンがあります。基本的には以下のようになります。

［1］オブジェクト名.メソッド
［2］オブジェクト名.プロパティ
［3］命令

オブジェクト名は省略できるものもあります。オブジェクトはウィンドウやドキュメント，画像などが該当します。

メソッドはオブジェクトに「指示をする」ものです。たとえばウィンドウを閉じる場合，window.close()となります。オブジェクト名とメソッド名は「.(ピリオド)」で区切ります。このピリオドの部分を「～を」「～の」と日本語で読み替えるとわかりやすいでしょう。

表1　オブジェクト

ウィンドウ	window
ドキュメント	document
フレーム	frame
日　付	Date
数学関数	Math
ブラウザ	navigator
ヒストリー	history
フォーム	form
画面	screen
画像	image
レイヤー	layer

「window」.「close()」
「ウィンドウ」を「閉じる」

メソッドはオブジェクト名の後ろに記述しメソッド名()というようにカッコで終わるので()だったらオブジェクトに指示を与えているのだな，とわかると思います。

メソッドはオブジェクトに指示を与えますが，オブジェクトの状態を示すものが「プロパティ」です。プロパティもオブジェクト名の後ろに記述しますが，カッコはつきません。たとえば文書の背景色を変更する場合は以下のようになります。

```
document.bgColor = "blue"
```

「文書」の「背景色」をblueにする

プロパティは読み込み，書き込みを行うことができますが，なかには読み込みしかできないものもあります。

JavaScriptの行末は「;(セミコロン)」または「改行」で区切られます。先ほどの背景色を変更するものは以下のようになります。

```
document.bgColor = "blue";
```

または，

```
document.bgColor = "blue"
```

どちらでも正常に動作します。

■変数

変数は「入れもの」です。JavaScriptでは変数には「なんでも入る」ようになっています。数値でも文字でもオブジェクトでもなんでも入ります。入れたものが数値であれば各種演算が可能になります。数値か文字かは自動的に判断し処理してくれます。

＊1　Mac版のExplorer 3，4はWindows版とは別ものです。補足できる部分については補足しておきましたが，動作が異なるため期待した結果にならないことがあります。また，Win版のExplorer 3でも確実な動作が保証できませんのでエラーが発生したり正常に動作しない(同じ機種，同じバージョンであっても)場合があります。

変数名は英数字で構成されアルファベットもしくはアンダーバーで始まります。日本語は残念ながら使えません。基本的に長さは自由ですがNetscape 2では1行の長さが255文字までという制限がありますので，240文字程度と考えたほうがよいでしょう(そこまで長い変数名はつけないと思いますが)。

変数名のアルファベットの大文字と小文字は判別されますので注意してください。abcとAbcは別々の変数として扱われます。

変数に値を代入するには，

```
変数名 = 値;
```

のように左側に変数名，右側に代入する値や文字列を書きます。たとえば，

```
[1] ohx = 1998;
[2] ohx = "1998";
```

のようになります。[1]の場合は数値が[2]の場合は文字列が変数ohxに入ります。

■演算

JavaScriptは四則演算，ビット演算などかなり細かい演算処理が可能です。以下の表に演算子を示します。算術子の中で+(プラス)のみ文字列にも適用することができます。たとえば

```
Str1 = "ABC"
Str2 = "DEF"
Str = Str1 + Str2
```

の場合変数StrにはABCDEFが入ります。

■数値，文字列，コメントなど

JavaScriptでは整数型，実数型という区別はなく数値であればなんでも扱うことができます。数値は10進数だけでなく8進数，16進数が使用できます。16進数のアルファベット(A,B,C,D,E,F)は大文字でも小文字でも使うことができます。

8進数：先頭に0を付加(例：018)

10進数：先頭に0を付加しない(例：1285)

16進数：先頭に0xを付加(例：0x1EF)

文字列は「"(ダブルクオーテーション)」または「'(クオート)」で囲みます。文字列数に制限はありません。*2

長いプログラムを作成する場合，スクリプト中にコメントを記述しておくとあとで見直すときに便利です。JavaScriptのコメントは「//」(スラッシュ2個)の後ろに記述します。コメントは行末までとなります。//だけでなく「/*で始まり*/で終わるまで(複数行にまたがることも可)」でもコメントとなります。

*2　Netscape 2.0xのみ1行の長さが255文字のため，255文字以上の文字列を扱う場合はmsg = msg + "sample"のように+演算子を使って文字列を連結するとよいでしょう。もっともNetscape 2.0xはほとんど使用されていないため無視できる範囲だと思います。

■関数定義

いくつかの命令を組み合わせた処理を関数として定義することができます。関数を定義する場合，<HEAD>～</HEAD>内に記述するのが普通です(図1)。

これはページの先頭からデータが読み込まれページが構築されるのですが，ページ構築中でもフォームのボタンやリンク，画像は表示されクリックなどを行うことができます。このときに関数が<BODY>～</BODY>にあり，ページが完全に読み込まれていない場合，未定義エラーとなってしまいます。このようなエラーを避けるため関数は<HEAD>～</HEAD>間に記述します。

図1　HTMLでの書き方

```
<HTML>
<HEAD>
<TITLE>Sample</TITLE>
<SCRIPT Language="JavaScript">
<!--
ここで関数を定義します。
// -->
</SCRIPT>
</HEAD>
<BODY>
本文
</BODY>
</HTML>
```

関数を定義するには以下のようになります。

```
function 関数名()
{
  定義内容
}
```

関数は同じ名前のものが再定義できます。この場合，あとから定義された関数が有効となり，定義前の同名の関数は消滅します。

■イベント

イベントはマウスボタンが押された，キーが押されたなど，なにか変化が発生した場合に通知されるものです。JavaScriptのプログラムの多くは，この「イベントの発生」により動作します。イベントはいくつも用意されていますが，ブラウザのバージョンやExplorerかNetscapeかによっても多少異なります。

主なイベントは以下のとおりです。

クリックされた	onClick
ダブルクリックされた	onDblClick
マウスポインタが上に載った	onMouseOver
マウスポインタが離れた	onMouseOut
マウスボタンが押された	onMouseDown
マウスボタンが離された	onMouseUp
キーが押された	onKeyDown
キーが押され続けた	onKeyPress
キーが離された	onKeyUp
ページが完全に読み込まれた	onLoad
ほかのページに移行した	onUnload

これらのイベントは「タグのオプション」として書きます。たとえばハイパーリンクを設定する<A>タグの場合，

```
<A HREF="../" onClick="alert('Oh!X')">
押して！</A>
```

のようになります。オプションは，

「イベント名」=「実行するJavaScriptの命令」

という形式になります。この場合，イベント名の大文字小文字は関係なく無視されます。

Explorer 4では，ほとんどのタグにイベントを記述することができますが，Netscapeでは，ほとんどのタグにイベントを記述できません。たとえば以下のHTMLはExplorer 4.0では画像をクリックするとダイアログが表示されますが，Netscapeではなにも起こりません。

```
<HTML>
<HEAD>
<TITLE>Sample</TITLE>
```

表2　演算子一覧

加算：＋	(例：z = x + y　→　16)
減算：－	(例：z = x－y　→　8)
乗算：＊	(例：z = x＊y　→　48)
除算：／	(例：z = x / y　→　3)
剰余：%	(例：z = x % y　→　0)
インクリメント：＋＋	(例：y = ++x　→　y=13，x=13：代入前に加算)
インクリメント：＋＋	(例：y = x++　→　y=12，x=13：代入後に加算)
デクリメント：－－	(例：y = －x　→　y=11，x=11：代入前に減算)
デクリメント：－－	(例：y = x－　→　y=12，x=11：代入後に減算)
符号反転：－	(例：y = －x　→　y = -12)
論理積：&	(例：15 & 9　→　9 (1111 & 1001 = 1001))
論理和：｜	(例：15｜9　→　15 (1111｜1001 = 1111))
排他的論理和：^	(例：15 ^ 9　→　6 (1111 ^ 1001 = 0110))
左シフト：<<	(例：9<<2　→　36 (1001<< = 100100))
符号付き右シフト：>>	(例：9>>2　→　2 (1001>> = 10))
符号なし右シフト：>>>	(例：19>>>2　→　4 (10011>>> = 100))

＊なお，x = 12, y=4とします。またビット演算，論理演算は32ビット長で行われます

＊NetscapeとExplorerでは演算精度が違い，Explorerのほうが精度はよくありません

```
</HEAD>
<BODY>
<IMG SRC="title.gif" onClick="alert('oh!X')">
</BODY>
</HTML>
```

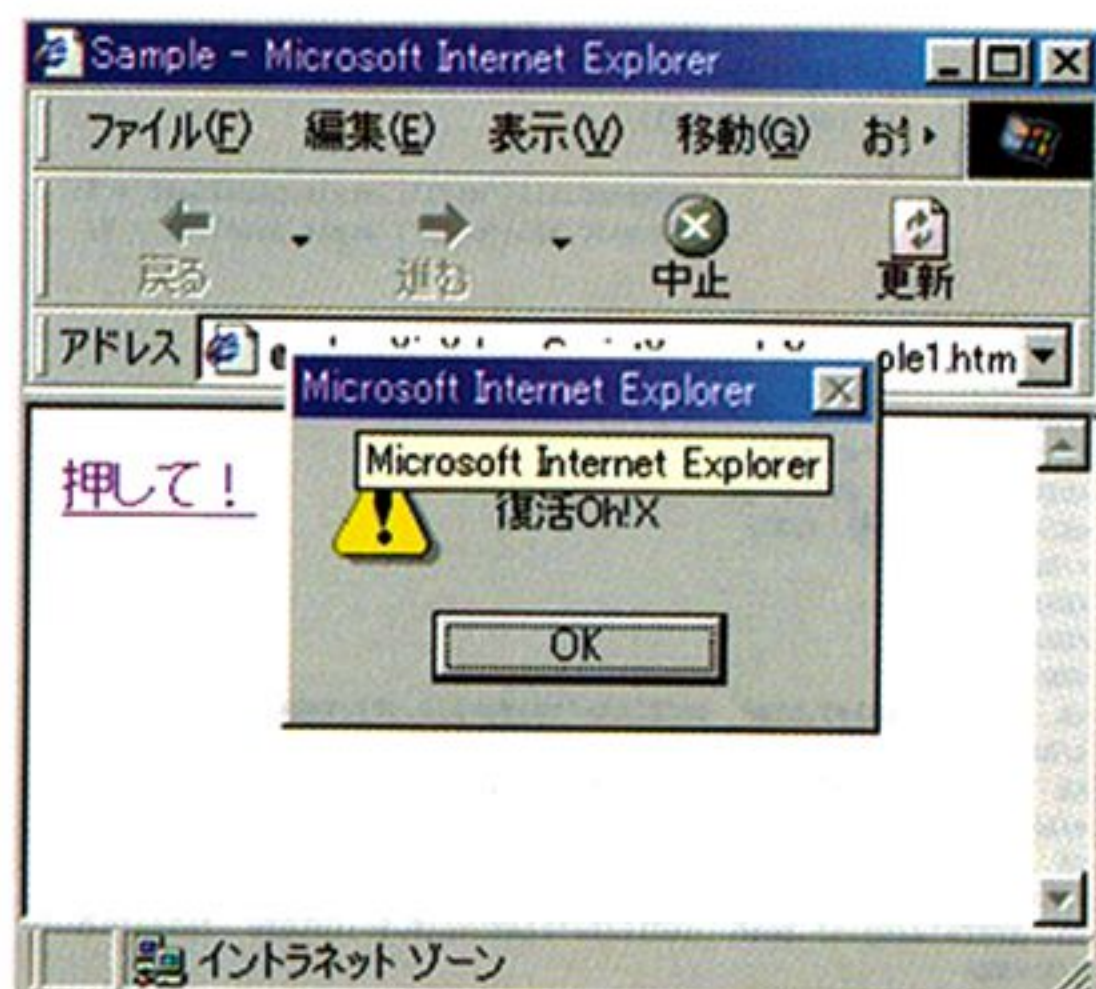
しかたないので共通で使用できるようにするには，<A>タグで囲むことになります。上記スクリプトは以下のようになります。

```
<HTML>
<HEAD>
<TITLE>Sample</TITLE>
</HEAD>
<BODY>
<A HREF="javaScript:void(0)" onClick=
"alert('oh!X')">
<IMG SRC="title.gif" BORDER="0">
</A>
</BODY>
</HTML>
```

とりあえず，最低限の基礎はこれだけです。あとはプロパティに値を設定して背景色を変化させたりしてみましょう。JavaScriptを使えば結構手軽にページに変化をつけることができます。

■ダイアログを出してメッセージを表示する(Sample1.htm)

まず手始めにハイパーリンク文字をクリックしたら「復活Oh!X」と表示させてみましょう。文字はダイアログを出して「復活Oh!X」の文字を表示させます。ハイパーリンクを設定するタグは「<A>」，クリックされたら動作するイベントは「onClick」です。alert命令はカッコ内の文字や値をダイアログに表示してくれます。

```
<HTML>
<HEAD>
<TITLE>Sample</TITLE>
</HEAD>
<BODY bgColor="white">
<A HREF="javaScript:void(0)" onClick=
"alert('復活Oh!X')">押して！</A>
</BODY>
</HTML>
```

sample1.htmの実行結果。ダイアログが開いた

ハイパーリンクタグのHREFの部分はスクリプトが書いてあるファイル名(sample1.htm)にしてあります。これはURLを指定しない場合，カレントディレクトリ内が表示されてしまうのを防ぐためです。*3

また，alert("表示")とすると文字が化けて表示されてしまうブラウザもあります(Netscape 4.04以前)。この場合は文字化けした文字の後ろに¥を入れると正常に表示されます。つまりalert("表¥示")とすればOKです。

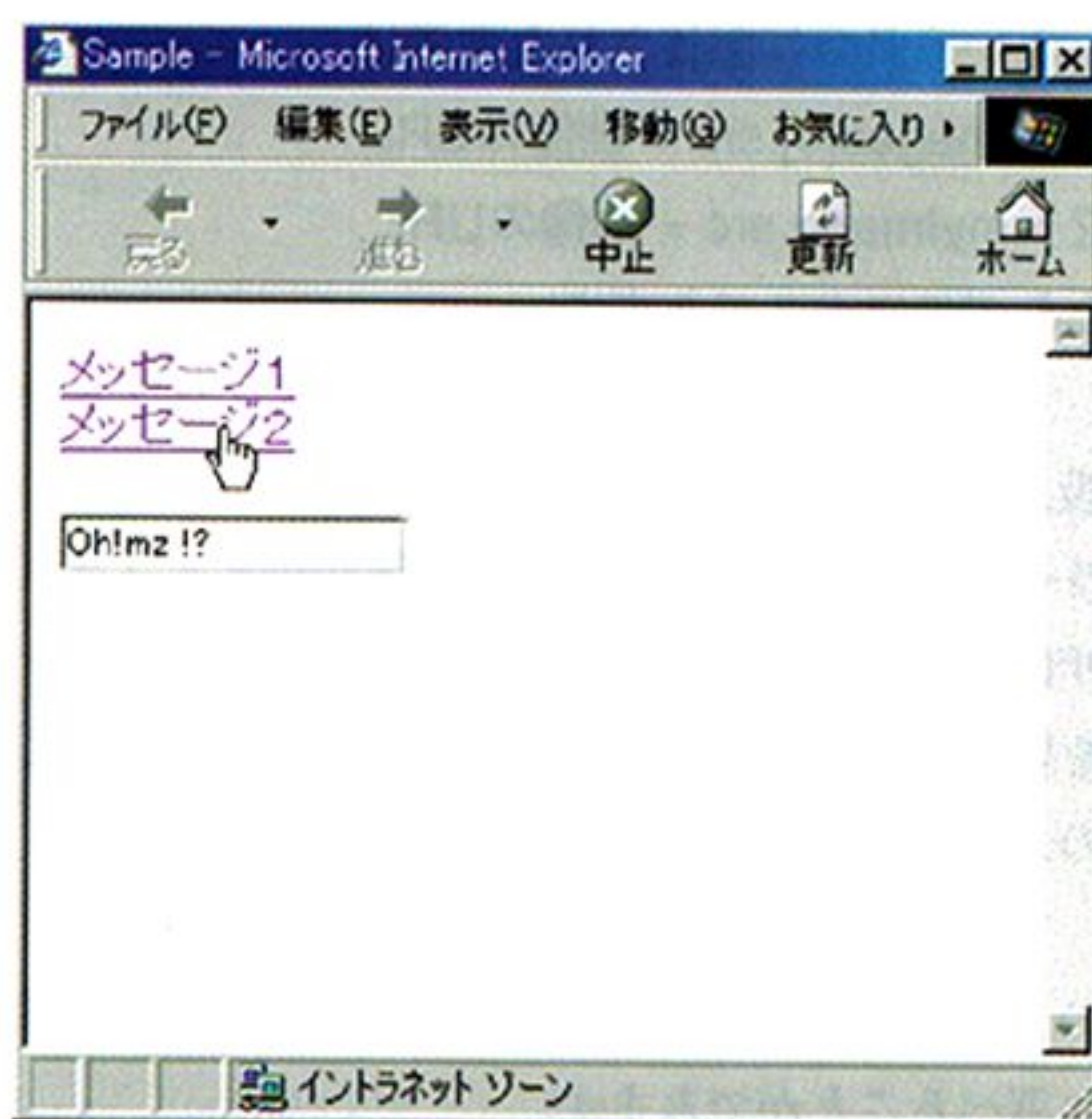
＊3　NetscapeではURLにjavaScript:void(0)と指定することでハイパーリンクさせないようにできます。便利ですので覚えておくとよいでしょう。

■フォームのテキストボックスにメッセージを表示する(sample2.htm)

今度はハイパーリンクの文字の上にマウスポインタが重なったらフォームのテキストボックスにメッセージを表示させましょう。フォームのテキストボックスに文字を表示するには「document.フォーム名.エレメント名.value = "表示文字"」とします。*4　フォームとテキストボックスに名前をつけるには<FORM>,<INPUT>タグのオプションであるNAMEを使います。NAMEで指定された名前が，そのままJavaScriptで指定するオブジェクトの名前になります。

マウスポインタが重なった場合はonMouseoverイベントが発生します。

```
<HTML>
<HEAD><TITLE>Sample</TITLE>
</HEAD>
<BODY bgColor="white">
<A HREF="sample2.htm" onMouseover
="document.myForm.disp.value='復活'">
メッセージ1 </A>
<BR>
<A HREF="sample2.htm" onMouseover
="document.myForm.disp.value='Oh!mz
!?'">メッセージ2
</A><BR>
<FORM NAME="myForm">
<INPUT TYPE="TEXT" NAME="disp">
</FORM
</BODY>
</HTML>
```

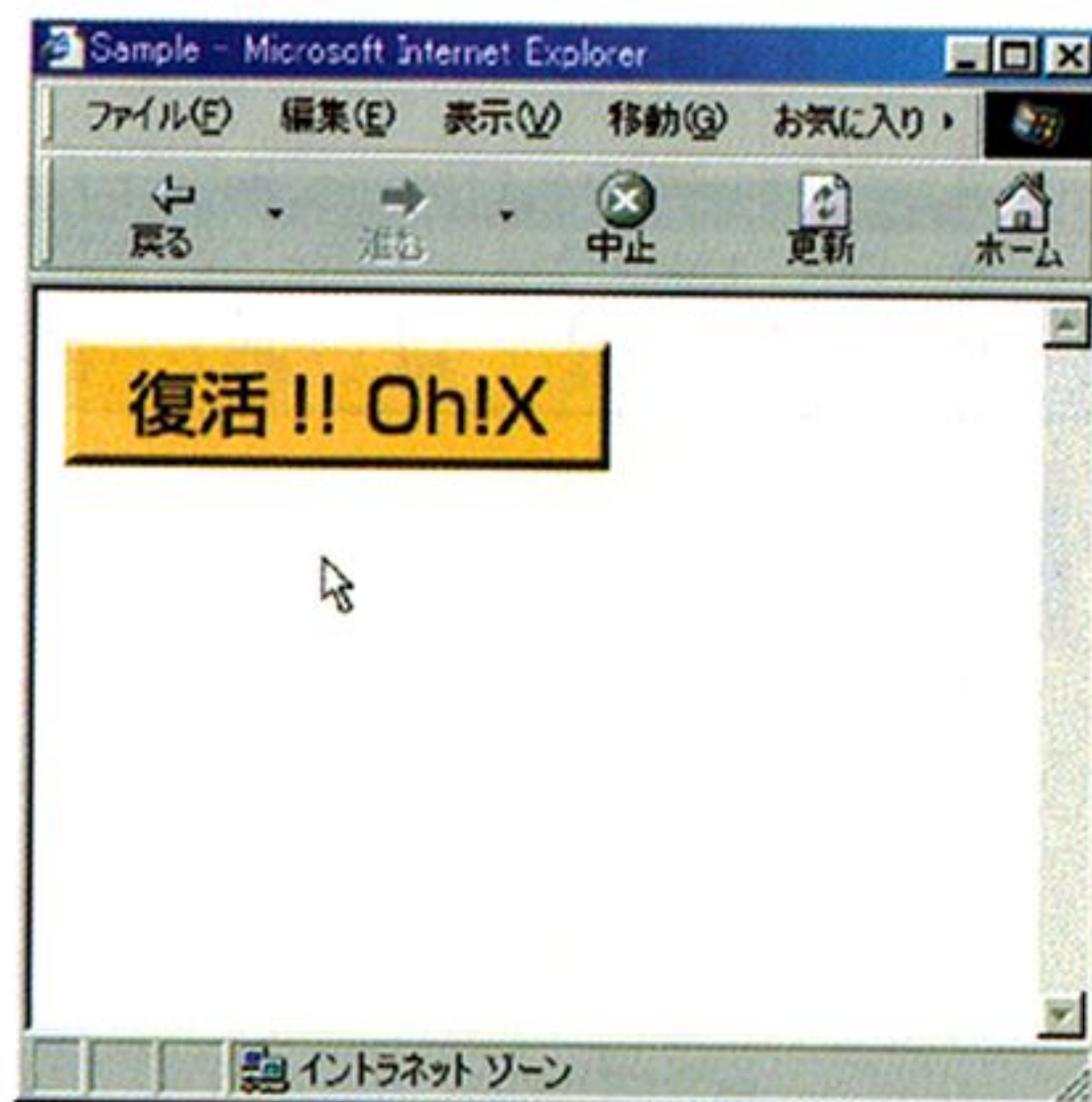
＊4　Netscape 2.0を除いて「フォーム名.エレメント名.value = "表示文字"」という書き方ができます。多くの場合Netscape 2.0のみ動作が異なることがあります。

sample2.htmの実行結果。選択に応じてメッセージを変える処理だ

■画像を入れ替える(sample3.htm)

画像を入れ替えるには，画像のオブジェクト「image」に格納されている画像のURLを設定します。画像のURLは「src」というプロパティに入っています。文書中の画像に対して以下の方法を使って制御することができます。

1）images[画像番号].src = 画像のURL
2）画像名.src = 画像のURL

画像番号はページが構築された順番に0，1，2……nという0から始まる連番になります。画像の名前は<IMG>タグのオプションである「name」を使って指定したものになります。たとえば，

```
<HTML>
<HEAD>
<TITLE>Sample</TITLE>
</HEAD>
<IMG SRC="abc.gif" name="myImage">
```

sample3.htmの実行結果。画像部分を押すたびに表示される画像が入れ替わる

という具合に記述された画像は,

1）images[0].src = 画像のURL

2）myImage.src = 画像のURL

としてアクセスできます。

またExplorer 3など画像関係のスクリプトを扱えないブラウザではmz.src='2.gif'とするとエラーが発生してしまいます。画像が扱えるブラウザはNetscape 3,4,Explorer 4ですがバージョンを判別してプログラムを作成すると大変です。このような場合は,

```
if(document.images)mz.src='2.gif'
```

のようにif文を使って画像が扱えるかどうか簡単に調べることができます。

```
<HTML>
<HEAD>
<TITLE>Sample</TITLE>
</HEAD>
<BODY bgColor="white">
<A HREF="sample3.htm" onMouseOver
="if(document.images)mz.src='2.gif'"
onMouseOut="if(document.images)mz.sr
c='1.gif'">
<IMG SRC="1.gif" BORDER="0" name="mz">
</A>
</BODY>
</HTML>
```

■背景色を変更する(sample4.htm)

背景色は「bgColor」プロパティにカラー名またはカラーコードを入れることによって変更することができます。カラー名はX Windowのものが使用できます(white, pin, tealなど)。またカラーコードはHTMLで使用する#RRGGBB(赤緑青2桁の16進数)とまったく同じものです。

```
<HTML>
<HEAD>
<TITLE>Sample</TITLE>
</HEAD>
<BODY bgColor="white">
<A HREF="sample4.htm" onMouseOver
="bgColor='red'">赤色にする</A><BR>
<A HREF="sample4.htm" onMouseOver
="bgColor='blue'">青色にする</A><BR>
<A HREF="sample4.htm" onMouseOver
="bgColor='yellow'">黄色にする</A><BR>
<A HREF="sample4.htm" onMouseOver
="bgColor='white'">白色にする</A><BR>
</A>
</BODY>
</HTML>
```

sample4.htmの実行結果。これは見てのとおりで，背景色を変更するものだ

■文字の位置を変更する(sample5.htm)

JavaScriptを使うと文字の属性や位置を変更できます。文字の位置を変更する場合はJavaScriptだけでなくHTMLおよびスタイルシートを設定する必要があります。まずHTMLのタグですが，Netscapeでは主に<LAYER>タグを使って実現します。これに対してExplorerでは<DIV>,<SPAN>タグなどを利用して実現します。しかし，このままではNetscape用とExplorer用，別々のHTMLを書かなければなりません。そこで両者で共通して使用できる<DIV>タグを使います。Explorerでは単純に<DIV>と記述すれば，タグで囲まれた部分の座標位置を変更することができます。これに対してNetscapeでは<DIV STYLE="position:absolute">といったようにポジションを指定しないとJavaScript側で参照することができません。そのため両者で共通に使う<DIV>タグは最低，以下のように書く必要があります。

```
<DIV id="名前" STYLE="position:位置指定方法">
文字……
</DIV>
```

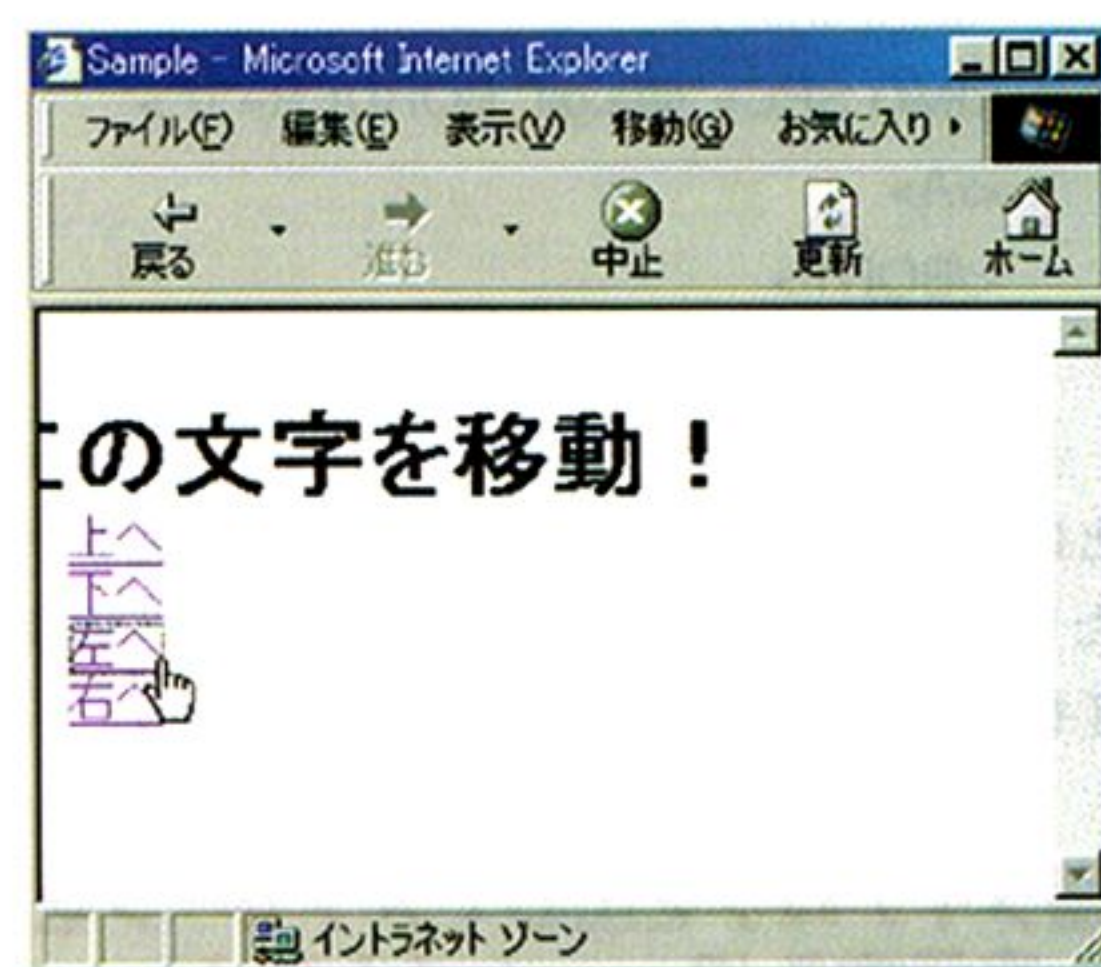
これでJavaScriptで<DIV>タグで囲まれた部分が参照できるようになりますが，NetscapeとExplorerでは参照するための命令が異なります。座標は以下のように参照します。レイヤー名とあるのは<DIV>タグのidオプションで指定した名前です。

- **Netscape**
  - 上：document.layers["レイヤー名"].top
  - 下：document.layers["レイヤー名"].bottom
  - 左：document.layers["レイヤー名"].left
  - 右：document.layers["レイヤー名"].right
- **Explorer**
  - 上：document.all("moji").style.pixelTop
  - 下：document.all("moji").style.pixelBottom
  - 左：document.all("moji").style.pixelLeft
  - 右：document.all("moji").style.pixelRight

このため位置を読み出す場合，書き込む場合NetscapeとExplorerを判断させて別々に処理させる必要があります。ブラウザのバージョンなどで判断する方法もありますが，手軽な方法はオブジェクトの有無を調べるものです。これは以下のように指定します。

- layerが使用できる(Netscape 4以上ということを示す)

  if (document.layers) 使用可能なときの処理
- allが使用できる(Explorer 4以上ということを示す)

  if (document.alls) 使用可能なときの処理

sample5.htmの実行結果。文字の表示位置を変更する。画面外の位置も指定できる

リスト1　sample5.htm

```
<HTML>
<HEAD>
<TITLE>Sample</TITLE>
<SCRIPT Language="JavaScript">
<!--
function theMove(dx,dy)
{
		X = 0;
		Y = 0;
//  現在の座標を読み出します
		if (document.layers)
		{
				X = document.layers["moji"].left;
				Y = document.layers["moji"].top;
		}
		if (document.all)
		{
				X = document.all("moji").style.pixelLeft;
				Y = document.all("moji").style.pixelTop;
		}
//  座標を加算します
		X = X + dx;
		Y = Y + dy;
//  移動後の座標を書き込みます
		if (document.layers)
		{
				document.layers["moji"].left = X;
				document.layers["moji"].top = Y;
		}
		if (document.all)
		{
				document.all("moji").style.pixelLeft = X;
				document.all("moji").style.pixelTop = Y;
		}
}
// -->
</SCRIPT>
</HEAD>
<BODY bgColor="white">
<DIV id="moji" STYLE="position:absolute">
<H1>この文字を移動！</H1>
</DIV>
<BR>
<BR>
<BR>
<A  HREF="sample5.htm"  onClick="theMove(0,-8);return  false">上へ
</A><BR>
<A  HREF="sample5.htm"  onClick="theMove(0,8);return  false">下へ
</A><BR>
<A  HREF="sample5.htm"  onClick="theMove(-8,0);return  false">左へ
</A><BR>
<A  HREF="sample5.htm"  onClick="theMove(8,0);return  false">右へ
</A><BR>
</BODY>
</HTML>
```

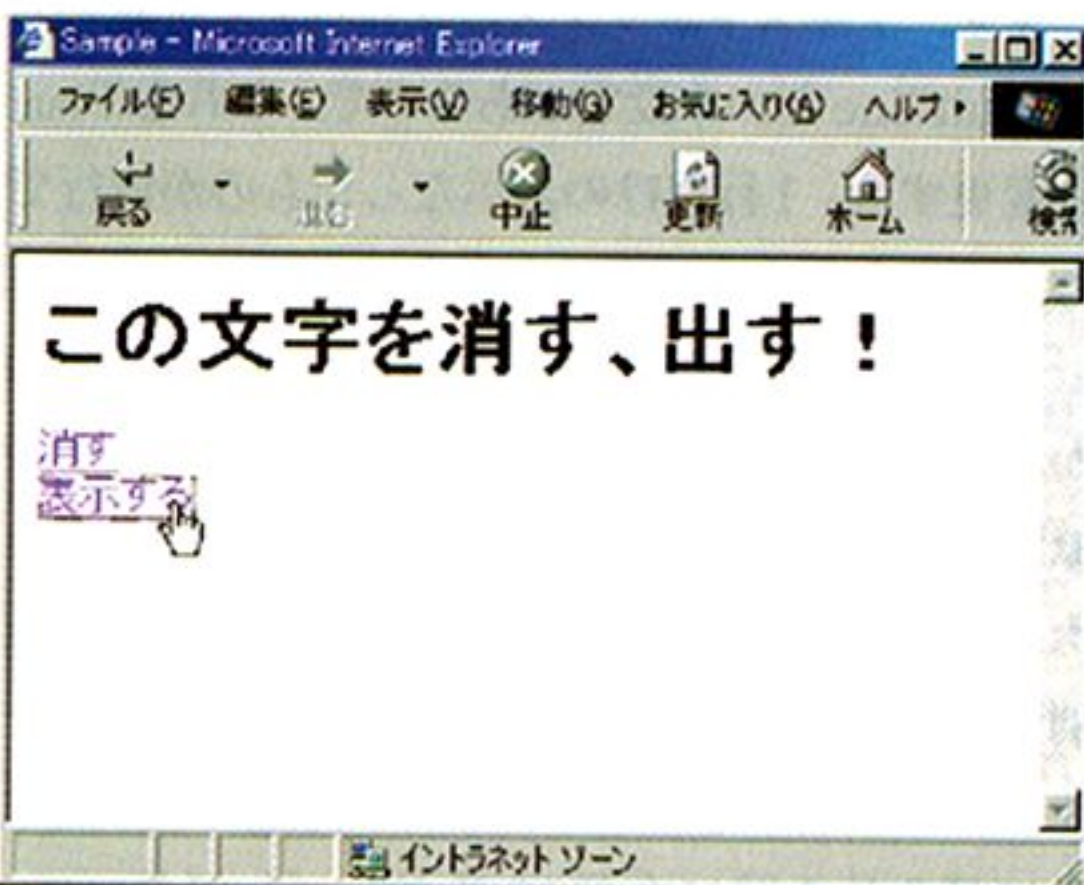
このようにすることで手軽に処理を分けることができ，昔のブラウザでもエラーを起こしません。

<A>タグでonClick="～;return false"と書かれていますが，このreturn falseはNetscapeでハイパーリンク指定してあるページへジャンプするのを防ぐためです。return trueとするとHREF="URL"で指定したページへジャンプします。

■文字を消す，出す（sample6.htm）

文字を表示したり消したりする場合は，visibilityプロパティを操作することで実現できます。visibilityプロパティに設定する文字はNetscapeとExplorerでは異なっています。

- Netscape
  - show または visible
  - hide または hidden
- Explorer
  - visible
  - hidden

両者で共通して使える文字はvisible，hiddenです。必ずダブルクオーテーションまたはクオーテーションで囲み文字列をvisibilityプロパティに設定してください。

sample6.htmの実行結果。ちょっとダイナミックコンテンツな感じで文字を表示できる

sample7.htmの実行結果。表示/非表示などを切り換える。画像の指定は互換性をとるのが面倒だ

■画像を消す，出す，入れ換える（sample7.htm）

<DIV>タグで囲まれた画像の内容を変更する場合は注意が必要です。というのもNetscapeとExplorerでは，入れ子になった画像の指定方法が異なるためです。Netscapeでは，

　document.layers[親のレイヤー名].
　document.画像名

として参照するのに対してExploerでは，

　document.画像名

と指定します。この方法を使ってもよいのですが，もっと手軽かつ安全に画像の場所を取得する方法（オブジェクト階層位置）があります。<IMG>タグにはonLoadという画像が読み込まれたときに発生するイベントがあり，このイベントが発生したときに「myImage = this」とすると変数myImageに画像の場所が入ります。あとはSample3と同様「画像名.src」，つまり「myImage.src」で画像を参照できるようになります。

Sample6では，レイヤーを指定するのにdocument.layers["レイヤー名"]，document.all("レイヤー名")のように指定していましたが，NetscapeとMac版のExplorerでは，

　document["レイヤー名"]

と指定することができます。Win版とMac版のExplorerでは，

　window["レイヤー名"]

と指定することができます。同じExplorerでもMac版とWin版は別ものに近いのでWin版で動くからMac版でも動く，Mac版で動くからWin版でも動くという保証はありません。

■現実的な問題

現実的な問題としてJavaScriptはNetscape，Explorerでの格差が多くあります。Explorer 3までは，結構互換性の高いものでしたがExploer 4，Netscape 4になってからは，相当数非互換部分が増えてしまいました。CGI，SSIの代用だったNetscape 2時代，ほかのオブジェクトと連携させるNetscape 3時代，そしてHTMLまで操作できるようになったNetscape 4/Explorer 4の時代。

両者にはなるべく非互換性の部分は埋めてもらいたいと思います。特にExplorer 4はWindows，Mac，UNIX版で別もの状態ですので結局Window 95/98/NTの人にしか見れないものができあがってしまいます。これでは，なんのためのインターネットなのかわからなくなってしまいMS-DOSの時代に逆戻りです。Netscapeもバージョン4になってから多くのバグが発生し期待どおり動作しないことが多々あります。

手軽に扱えるのがJavaScriptのよいところですが，それを殺さないようにしてほしいものです。

リスト2　sample6.htm

```
<HTML>
<HEAD>
<TITLE>Sample</TITLE>
<SCRIPT Language="JavaScript">
<!--
function theShowHide(types)
{
          if (document.layers) document.layers["moji"].visibility = types;
          if (document.all) document.all("moji").style.visibility = types;
}
// -->
</SCRIPT>
</HEAD>
<BODY bgColor="white">
<DIV id="moji" style="position:absolute;">
<H1>この文字を消す、出す！</H1>
</DIV>
<BR>
<BR>
<BR>
<A HREF="sample6.htm" onClick="theShowHide('hidden');return false">消す</A><BR>
<A HREF="sample6.htm" onClick="theShowHide('visible');return false">表示する</A><BR>
</BODY>
</HTML>
```

リスト3　sample7.htm

```
<HTML>
<HEAD>
<TITLE>Sample</TITLE>
<SCRIPT Language="JavaScript">
<!--
function theShowHide(types)
{
          if (document.layers) document["moji"].visibility = types;
          if (document.all) window["moji"].style.visibility = types;
}
// -->
</SCRIPT>
</HEAD>
<BODY bgColor="white">
<DIV id="moji" style="position:absolute;">
<IMG SRC="1.gif" name="img" onLoad="myImage=this">
</DIV>
<BR>
<BR>
<BR>
<A HREF="sample7.htm" onClick="theShowHide('hidden');return false">消す</A><BR>
<A HREF="sample7.htm" onClick="theShowHide('visible');return false">表示する</A><BR>
<A HREF="sample7.htm" onClick="if(document.images)myImage.src='1.gif';return false">画像その1にする</A><BR>
<A HREF="sample7.htm" onClick="if(document.images)myImage.src='2.gif';return false">画像その2にする</A><BR>
<BR>
<IMG SRC="1.gif">と<IMG SRC="2.gif">を入れ替え、表示、非表示にします。
</BODY>
</HTML>
```

VBでツールを作る

中野修一/Nakano Shuichi

またまたLevel1だけど，今度はちょっとだけ実践編だ。具体的なプログラムを通して，コーディングの基礎というよりはむしろ作り方や考え方の流れを解説していく。CD-ROMのサンプルを見ながら読んでみてほしい。

■ファイルダンプをしてみよう

前の記事で一応コントロールを作成してみたわけだが，たとえ動作の意味がわからなくても，肝心のイベントの作り方とプロパティの変え方はなんとなく理解してもらえたかと思う。

ではなにか少し実のあるものを作ってみよう。プログラムの練習はボタンやテキストをフォームに置いてちょこちょこいじっていても絶対に前進しない。具体的なモノを作っていくのがいちばんだ。さらにいえば，人が作ったモノを改造していくのが最良だと考えている(変にハマることもあるのだが，それも勉強のうちか？)。今回は2つのサンプルを用意した。どちらも作りかけというのが我ながら情けないような気もするが，用途を考えるとこれはこれでよいのだと思っている。搭載予定で未実装の部分のボタンなどを残してあるだけなので，最低限動く部分についてはほぼ問題はないはずだ。

まずなにを作るかが最初の問題だ。なにがほしいかを考える。Windows環境に移ってきてまず困ったのはダンプツールがないことだ。例によって，vectorや窓の杜で探してはみたが……ま，いいか。探せばいいのもどっかに落ちていることは間違いないのだが，練習用にはちょうどよい課題だろう。

まず，必要な機能と操作性を考える。

もっとも基本になる16進表示とASCIIダンプはまあどうにでもなるだろう。実用面で考えると，データは数値としても表示されないとわかりづらいことも多い。数値表現は必須として，文字コードはASCII形式でないこともある。JISコードやEUCコードは変換が面倒そうなのでWebブラウザにでも叩き込めばいいということにしても，文字コードでデータを詰めて入れてたり，オフセットがついてたりという変則的なものもたまにある。ということで，文字コードのオフセットにも対応しよう(なんに使うんだ？)。あとは，よくある(？)タグやチャンクを追いやすいようにオフセットジャンプやディスプレースメントジャンプがあるとデータやコードを見るのが楽だ。マーキングは複数の場所に設定できないと不便だし，さくさく動くことが基本。データ検索は必須。ファイルの比較とかもあると便利だろうけど，機能そのものよりもインタフェイスを作るのが面倒そうだ(ありがち)。これはFCコマンドで足ることにしておこう。ファイルエディットは当面できなくてもいいや……。

というふうに，だいたいの構想を胸にデザインを進めてみた。VBのプログラムはフォームのデザインから始まる。あとで変更するのも可能だが，最初にある程度構想を持って全体を設計することが重要だ。これはコーディングなどよりずっと大事なことだ。しかし，ロジカルなデッサン力がついてくるのは，プログラムの数をこなしたあとの話なので，仕様決定はこんな感じなんだと流して眺めておいてほしい。

さて，サンプルはCD-ROMに入っているが，ウィンドウはタイトに作るのが私のスタイルなので，ごちゃごちゃしてるし，未実装の機能もたくさんあるのでサンプルとしてはあまり適してないかもしれない。ツールというのは往々にして拡張予定の塊みたいなものだし，私用ツールってのはこんなもんだということにしよう。自分の使いやすいように作るというのが基本だ。他人にも使いやすければ理想的だが，だからといって，Windowsの作法に則ってプログラミングしましょうなどとやるつもりは毛頭ないので悪しからず。

■コントロールの動作を知る

ここでデザインしつつ，各コントロールがどんな挙動をするのかを把握していく。

まず，テキストボックス。プロパティを眺めるとMultilineという設定があるので，これをTrueに変更する。1行だけの表示だったものが何行でも表示できるようになったことがわかる。

いろいろ試すと文字が綺麗に並ばないことがわかる。文字の乱れはプロポーショナルフォントが原因だ。そこで，使用フォントは非プロポーショナルフォントに設定する。用途からして1とlの区別がつきやすいMS 明朝を基本とした。スクロールバーの設定もあるのでちょっと検討する。要するに読み込み時に全部16進ダンプしておいて，それをひたすら眺めるという手だ。メモリ上にテキスト化して置いておけば処理は非常に楽だ。が，数MBのファイルを開いたときのことを考えると，表示には約4倍のメモリ量が必要になるので，実用的ではない。表示まで待たされるのもイヤだし，そもそもテキストの容量制限があるようだったので，どうせそのままでは使えない。行数は表示分だけとして，必要な部分だけを表示させることになる。

とにかく，このように，やりたいことと用意されたコントロールなどとの折り合いがつくように仕様変更を繰り返していく。VBのシステムも結構膨大な量があるのですべてを把握しろというのも無理がある。やはり試行錯誤しながら基本設計を進めていくことになるだろう。探せば抜け道もあったりするので経験が必要な部分だが，私などはまだまだ経験が足りないのでかなり苦労している。

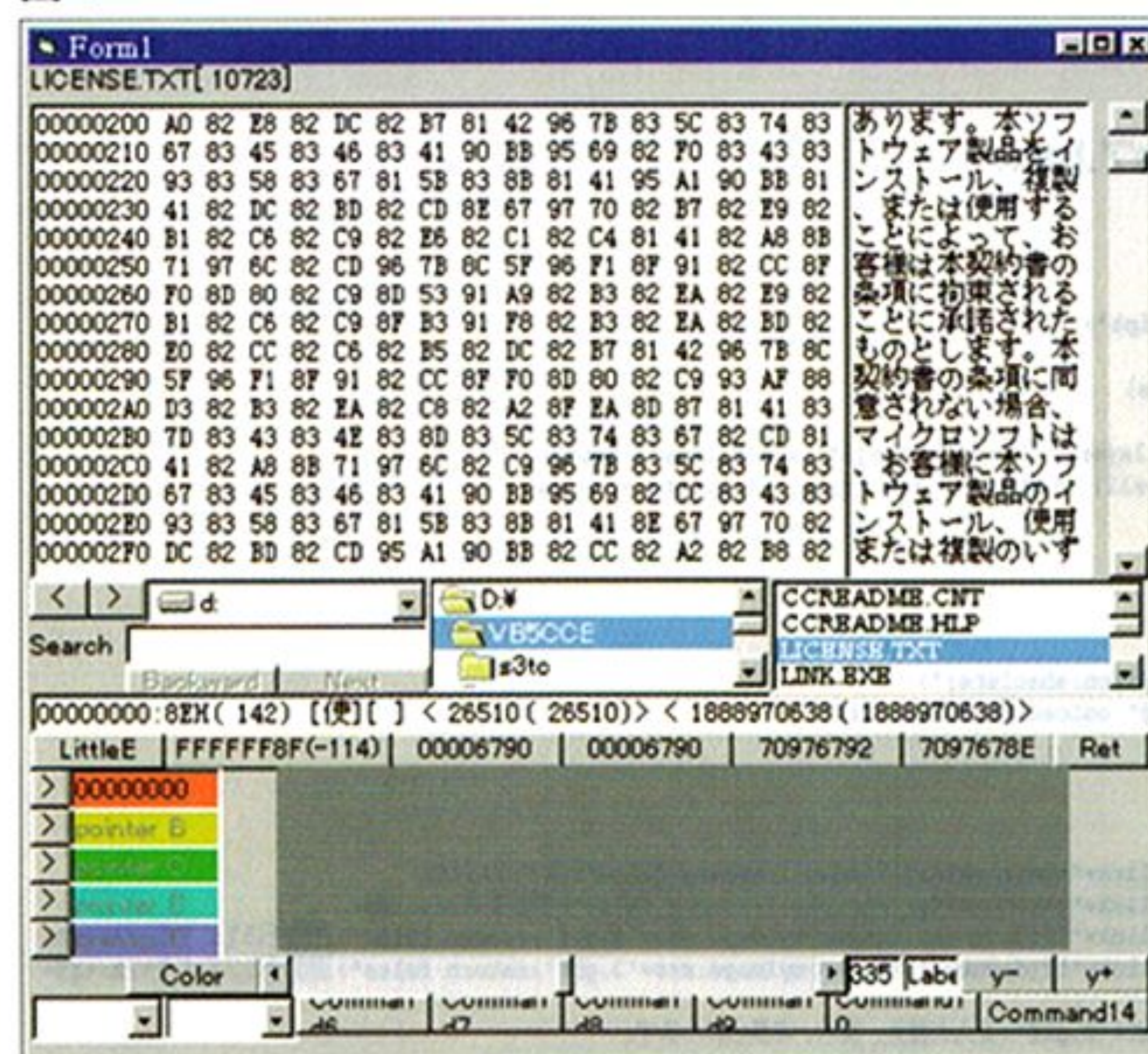
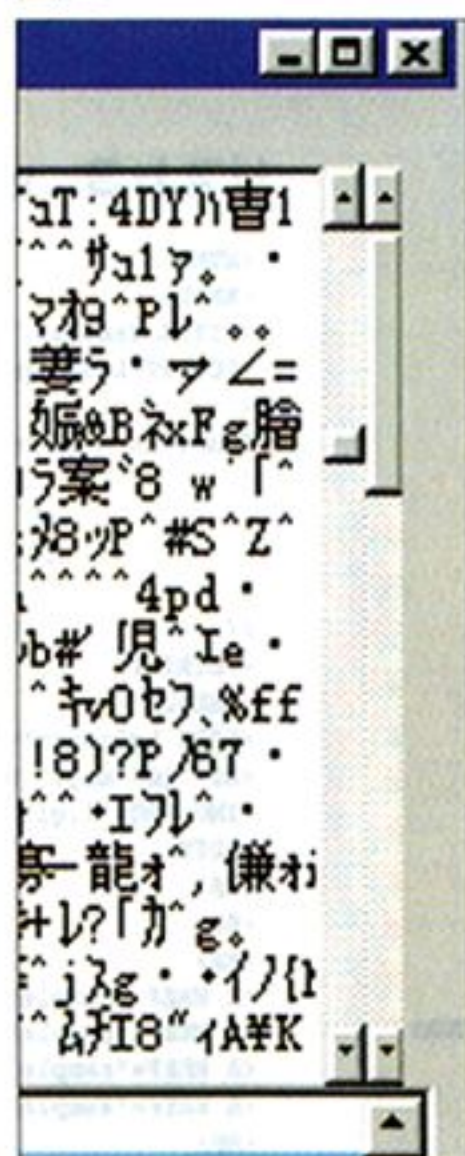
図1

単にファイルの中身を見るだけだが，手軽に扱えるツールがなぜかない

図2

これが2段分離スクロールバー。大きなファイルだと自動的に分離する

いくつかポイントを挙げてみよう。

第一関門がスクロールバーだ。

もともとの元祖VBとWindowsが16ビット仕様なので，-32768～32767までの範囲しか指定できない。1行16バイトとして1MBまでのファイルしか扱えない。ああ，なんて86な仕様。ページ単位のスクロールにすると16MBまで使えるが，昨今では20MBを超えるファイルもさほど珍しくはない。ファイルダンプツールとしてそれはまずいので，新しくスクロールバーを作った。

Control Creation Editionというだけあって（ちょっと意味が違う気もするが），VBではユーザーコントロールを作ることができるので，気に入らないコントロールは代わりになるものを作ってやればいい（同じ名前で再定義……というのはできなくて正解なのだろうなあ）。今回はちょっと手抜きして普通のスクロールバーを流用しつつ作成している。ファイルサイズが1MB以内だと普通のスクロールバーと変わりないのだが，それを超える上限サイズが指定されるといきなりスクロールバーが2本になる。ちょっと唐突だが，まあ使えなくはない。

とにかく，これまで使っていたスクロールバーへの指定をintegerからlongに変えるだけでほぼ同じ動作を期待できるようにしておくのがよいだろう。ついでにいうと，変数宣言一般でケチってintegerなんて使うとロクなことはないので，できるだけlongを使うようにしよう（それともintegerが32ビットでないとハマるのは68系のユーザーだけだろうか……）。部品を組み合わせていくことがVBの基本だから，部品を作ってやればいい。CCEで一度作っておけば，ほかのプログラムでもボタンなどのコントロールと同等に使うことができるようになる。

■プログラム解説

プログラムはゴチャゴチャしているので，わかりにくいかもしれないが，コーディング自体はたいしたことはしていない。それぞれのルーチンを読むのはさほど難しくないと思う。

意外と苦労しているのが32ビットアドレスの表示。符号なしの整数がないので，16進数にしたり演算したりするのにオーバーフロー対策で苦労している。結局文字列処理に落ち着いている部分もある。

ASCIIダンプでは，文字列幅が読めないので（全角文字も半角文字も同じ1文字として換算される），シフトJISコードで全角文字か半角文字になるのかをいちいちチェックして横幅を決めている。シフトJISのコードを知らないと手が出ない部分ではある。資料はどこかに転がっているものなので，手早く見つけることが肝心だ。配列の規則さえわかればなんということはない。

実行するとボタンの上にアドレスがたくさん出てくるのが確認できるだろう。さっきちらっと書いたが，なんのことだかわからない人にはわからないので，軽く説明しておく。データというのは直線的に流れていくだけではない。途中でブロックになっていたり，分岐したりする。プログラムだと16ビットくらいのディスプレースメント，グラフィックデータなどでは32ビットのオフセットをとるものが多い。IFFのチャンク構造も32ビット長だったなあ。チャンクの基本は，データを記述するとき，まずデータの種類を4バイトの文字列（識別子）で入れておき，次にそのブロックのデータサイズを入れておく。それだけ。目的のデータまで先頭からサーチしていくのもさほど苦にならないし，データの追加変更が柔軟に行えるほか，識別子はテキストなので，ダンプして見ていても人間にわかりやすいというフォーマットだ。データの内部自体は好きなように処理してかまわないのだが，同じルールでサブチャンクを作ってあることも多い。たぶん，IFFが元祖だと思うのだが，多くの構造的データではこのチャンク形式が用いられている。Windowsでも多く使われている。QuickTimeとかだと32ビットの長さが先にきて，次に4バイトの識別子がきてという感じでちょっとひねくれているのだが。

とにかく，メジャーなアドレッシングモードや32ビットオフセットを追いかけることのできる機構はデータ解析のためには必須だ。ということで，仕様に加えているわけだ。ほかにもいろいろこの手のものは出てくるだろう。基本部分さえ押さえておけばそのたびに拡張すれば済む（その後，32ビット実アドレスへのジャンプを加えた）。

なお，エンディアン（バイトの並び順）を変更できるのは当たり前の機能だろう。

ほかに特徴的な部分はグラフィック表示だろうか。ポインタからのデータがビットマップデータだと仮定して表示を試みる。グラフィックフォーマットにもいろいろあるので，ちゃんと対応していくと結構大変な作業になる。必要になったら拡張だ。ラスターデータへも対応する予定だが，手ごろなサンプルがないので全然実装していない。

BASICだと，データの構造解析を行いつつ，処理ルーチンを組み立てていくといったことが簡単にできるので，下手に実行ファイルにせずに，必要になったときに拡張できるインタプリタの形式のほうが実際には使いやすいことが多い。「ツール」として使うよりも，BASICというのは環境としての意味が大きいものだった。これは昔から変わらないし，私がBASICにこだわるのもそういった必要なときにすぐ試行錯誤できる環境がないと不便だからだ。

しかし私の場合，ざっとCのソースを書いてビルド……という真似がなかなかできないのだ。ということで，VBも実行ファイルを作るのを目的とせずにいろいろいじれる環境という意味で，VBごと起動して使うというのが正しいだろう（これで，常に起動しておける気になるくらいの安定感があれば……なあ。やたら重いし）。

■パターンエディタ

サンプルをもうひとつ。またまた個人的な用途で恐縮だが，アイコンとかいじれないというのが，

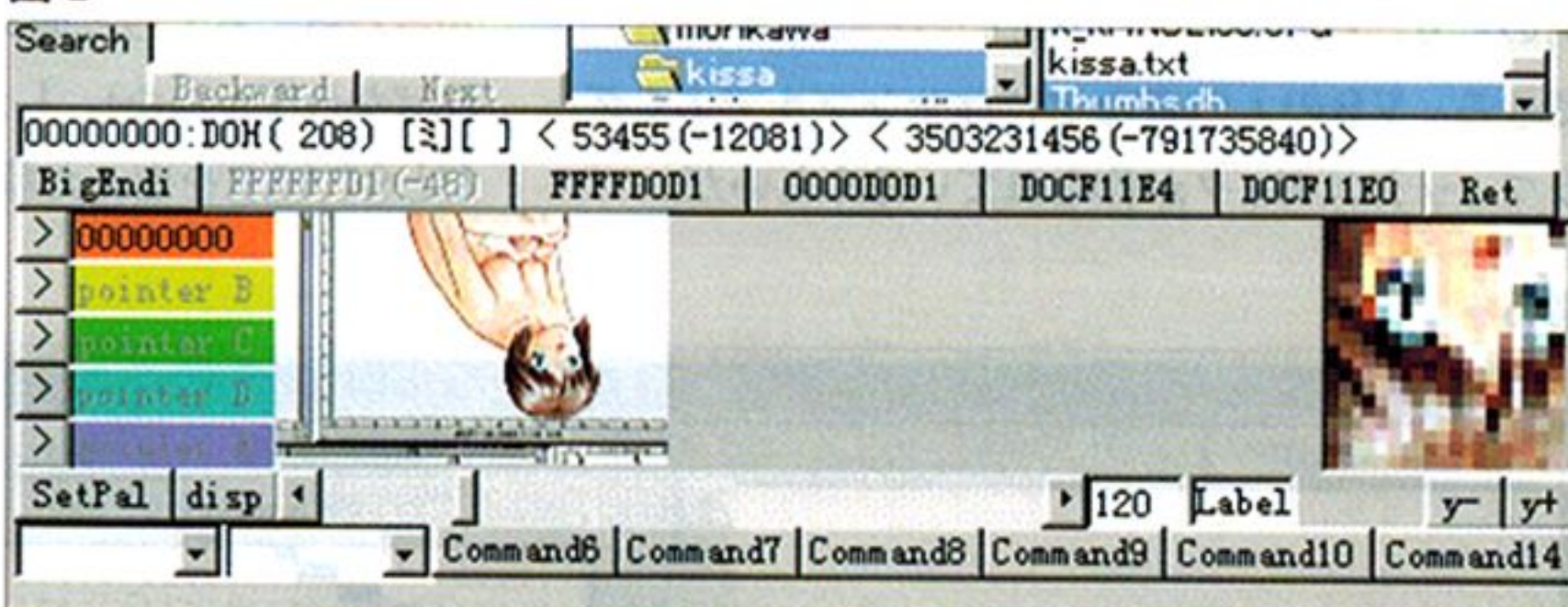
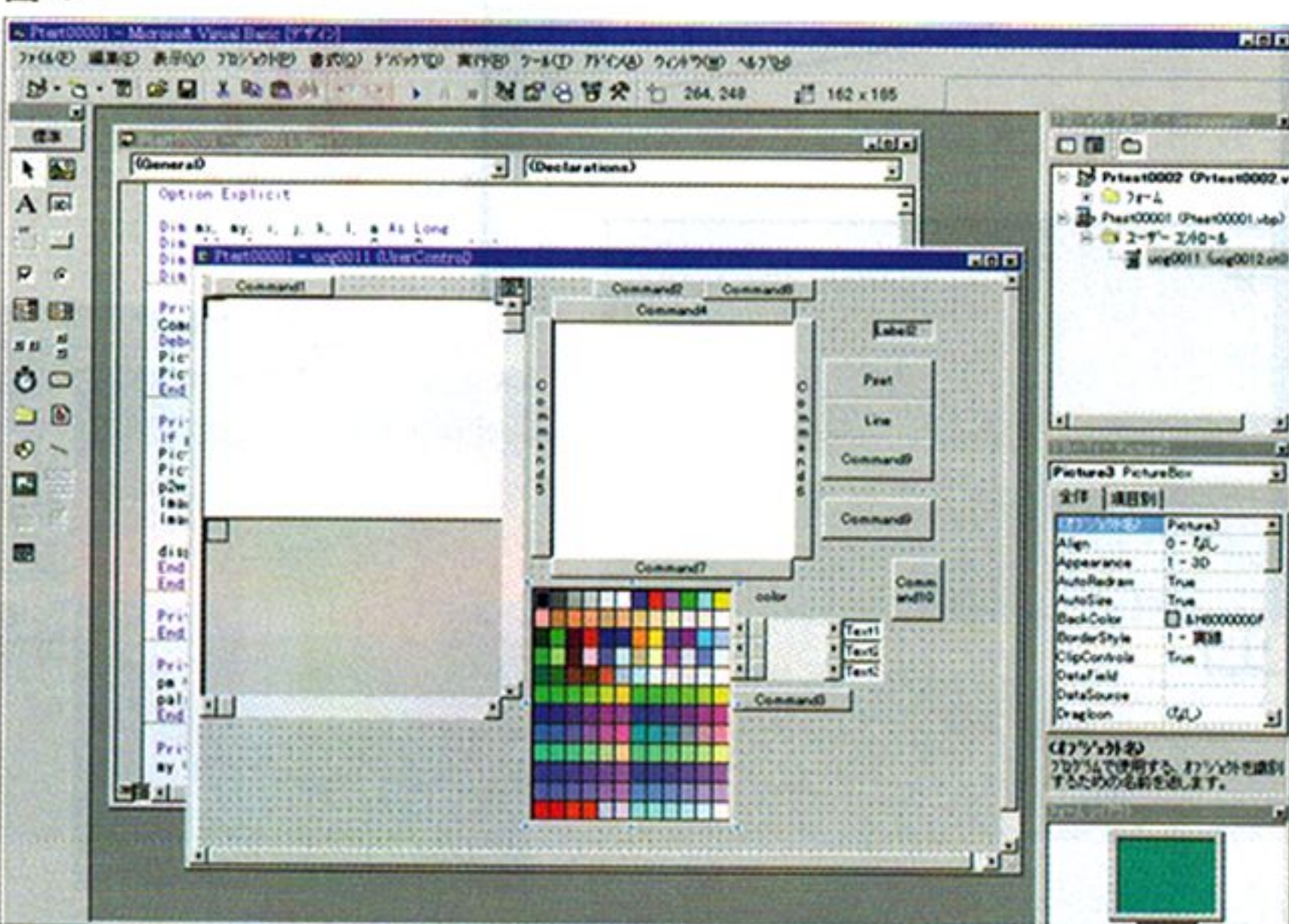
図3

グラフィックを表示したところ。これはIE4のサムネイル画像。Windowsだから上下が引っ繰り返っている。16ビットベタ画像だということがわかる

図4

パターンエディタを作る。とりあえず，全体的な雰囲気を決めてパーツを配置する。使わないものもたくさん並んでいる

Windowsがつまらない理由のひとつかもしれないなあと思っている。MSの標準ペイントって手があるのだが……。

最近はゲームなどのグラフィックでもドットを意識しないものが大半になってきている。実際，一般のグラフィックエディタで処理するという手もないではない。データはフルカラーで作っておけばシステムがうまく処理してくれる。が，根が古い人間なので，ドットを打つというのが必須機能のように思えてならないこともある。限定された色数での描画などもすでに過去の遺物かもしれないなあ，と思いいつ，ドットの打てる環境がほしくなってきたのだ。ドット打ちのためにはそれなりに特殊な操作性も要求されてくる。で，作ることにする。もちろん，探せばどこかにいいものはある（以下略）。

まず，フォームの設計からだ。今度はPictureBoxが主体だ。メイン画面は全体図と拡大図（エディットエリア）で構成する。あとはパレットやツールボックスだ。PictureBoxは裏画面用に1枚余分に取ってあるが，現在のところまったく使用していない。

操作自体は拡大図に行って全体図でも表示するという形をとる。がデータ的にはむしろ全体図に書き込んで拡大図で見る感じになる。

さて，Windowsにはさまざまな描画機能が用意されている。PictureBoxでも指定すれば線を引いたり，ペイントしたりといったことが簡単にできる。VBの入門書などにはサンプルでよくグラフィックエディタが作られているくらいだ。

が，グラフィック処理をちゃんとやっていこうとするとそれらは結局，使えないものだということがわかるだろう。マスク処理はどうするか，合成などに応用するには……などと考えると，基本処理を作り直すしかないことがわかる。

まず，点を打つ。

すべてのグラフィック処理の基本はこれに尽きる。

拡大部分はBOXFILLで実現している。

そして現在ではあまり意味がなくなった256色分のパレットを置いている。256色データはまだまだ多用されているのだが，パレット付きなので，扱いは実質フルカラーとほぼ変わりない。固定パレット256色などはナンセンスだ。……といいつつ，固定パレット256色での可能性をもの凄く意識したパレット構成になっているのは，わかる人にはわかるかもしれない。

256色エディタだと，パレットの並びが色を選びにくいものが多い。あまり意味がないこだわりだが，パレットパターンを3種類変えられるようになっている。その割には，選べない色とかあるのだが，とりあえず本人は気にしていない（バグはわかってるんだけどね）。

点の次は線だ。ラインを引く。

指定の最中につかんだラインがグリグリ動くのは当然だろう。まあ，これはXORを使うだけだからなんでもないことだ。幸いにも，Windowsの描画モードでそれは用意されている（されてなかったら，ちょっと面倒なことをして実装しなければならなかった）。しかし，問題はVBのラインだとSCALEを変えた状態の拡大画面では線が細いままだということだ。ドット位置計算時の座標系はスケーリングされるのだが，描画時の指定ではスケーリングは考慮されない。

これでは困る。こういうツールではだいたいの位置を知りたいのではなくて，どのドットが着色されるのかがわからないとラインなんて引けない。ということで，やはり作る。ベタなラインルーチンだが，それでも動いているからよしとしよう。コードの簡単さだけを求めたようなルーチンだ。

ちゃんとしたアルゴリズムでやるほうがよいのは当然だが，VBでは高速化しても限度がある。ちゃんとアセンブラで書く人はこういうのだけは参考にしないように。ちゃんとBuresenhamのアルゴリズムでラインの両端からダブルステップで書くように。ループは展開するなりラインサイズに収めるなり効率のよいのにするのは当然だ。ま，ここはBASICだから簡単に済ませている。一応使いものになる速度では動いてくれるし。

■今後の展開

今後はシードフィルとかいろいろ機能を加えていって，一人前のエディタにしてあげたいものだ。なーんもついてないので，ざっと触ってみて面白そうだと思った人は独自に拡張してみるのがいいだろう。グラフィック処理を基本から作っていくというのは，それなりに面白い。やるからにはいろいろ機能をつけたいし，SXのパターンエディタくらいの操作性があれば使う気にもなるだろう。速度さえ許せば普通のグラフィックエディタへの展開もなくはない（無理か？）。Windowsのアイコン直に書き換えるとかいろいろやりたいことはある。画像処理とかの実験台としてもよいベースかなあという気がしている。そのうち画像処理を解説していくこともあるだろうし，私がやるのだったら，BASICが基本なのは間違いないし……。

図5

実行例。ラインの処理や変なパレットだけしかついてない

図6

パレットの配置例その2。使いやすい配置を決めるために実験的に3パターンの配列を用意してみた。色はテーブルを基にプログラムで作成

Oh!X68000

Oh!Xがなくなってから，X68000関係の情報は
ネットワークか電脳倶楽部くらいでしか入らなくなってしまっていた。
ここでは主に満開製作所を中心とした，「その後のX68000」の状況を語ってもらう。
X68000はまだまだ死んでいない。
空白の3年間の総まとめから，X68000シリーズはこれからどこにいくのかを見定めよう。

その後のX68000
――現況そして展望――

あいはらてつや

あれから3年

Oh!X休刊号で，マイナーから伝説へ進化したとまでいわれたX68000シリーズ。それから3年経とうとする現在も，ずっと使い続けているユーザーにとって，現役のマシンであることに変わりはありません。

ただ，シャープ撤退の結果として残され，放置されてしまったハード的，ソフト的な弱点をどのような形でフィックスしていくか，ここが今後使い続けていくうえでの最大の問題といえるでしょう。

ということで，以下では「Oh!X休刊からこれまでのX68000の状況」について概括していきます。また，本体が買えないという「終わった」状況を，いかに乗り越えようとするのか，という点も紹介していくことにしましょう。現在までX68000をプラットフォームとして活動している企業は，唯一満開製作所しかありません。今回のこのコーナーの原稿は全体として満開製作所の人間が書いていますので，その辺が中心になってしまいますが，それはご容赦ください。

まず，満開製作所はまったくの零細企業です。つまり，それは自由に(主に金銭面で)開発研究する余裕がないということであり，作ったもののほとんどをちゃんと売らないと次の開発ができない，ということを意味しています。また開発研究しながらお客さんからの受注・発送までやる，などというのは，同一の人間の仕事としては手に余るものです。が，そうせざるをえないような，なかなかシビアな状況に置かれているのです。愛を育むのにもお金が必要なように，気合と根性だけではどうにもならないことは皆さんが思い浮かべるよりずっと多いのです。

本来先に立てるべき，今後のX68000をどう発展させていくか，といったコンセプトも，立てにくいだけでなく，立てても思うようにはコトが運ばない，というのが常なのです。そういう状況のなかで，これまでX68000に足りなかった点を少しずつ埋めてきたというのがこの3年近くの歩みだったといえるでしょう。

ぶっちゃけた話，最初から系統的に問題点を解消してきたわけではなく，やっていたらこうなった，というのが実情でもありますが……。

X68000に足りないもの

それでは特にX68000に足りないこと，問題になっていた点を挙げていきましょう。

- CD-ROMへの対応
- 新たな周辺機器への対応
- ハードの高速化・大容量化
- WWWへの対応
- OSのネットワーク対応

といったところが挙げられます。

当然，このほかにも，いまや，よそと見比べて狭くなってしまったグラフィック周りをどうするんだ，という声もあるかと思いますが，ユーザーが現状で希望するスペックを満たすには，かなり敷居は高いでしょう。技術的にそれほど困難ではありませんが，現状のマシンにPCIバスを載せて環境を構築する，今後満開製作所からリリースされるであろう「X68000互換機」を待つ，この二択になるでしょう。それらは今後の開発の動向に大いに期待してもらいたいところです。

●CD-ROMへの対応

実際，上記のうちCD-ROMと周辺機器への対

応，そして大容量化に関しては比較的早く(十分遅かったのですが)対応してきました。

CD-ROM環境での最大の問題はメーカー標準のドライバが最後まで存在しなかったということでした。現在は，GORRY氏作の汎用のSCSIドライバ「SUSIE」が満開製作所のおすすめする標準ドライバとなっています。このドライバは，先に常駐しているSCSIドライバを置き換える，あとからでも個別のドライブに対して設定できるなどの柔軟な特徴があります。

また，単なるCD-ROMドライバではなく，HD，MO，PD，ZIPなどのメディアにも対応しており，LUNの変更など，非常に柔軟にSCSIへアクセスする手段を提供しています。

●新たな周辺機器への対応

また，SUSIEと同時に使うことで，さまざまな新しい周辺機器に対応するためのドライバなどが作成されていることは見逃せません。SCSI接続タイプのLS-120ドライブの接続や，DVD-RAMドライブへの対応などです。その他，ドライバとは直接関わらない話ですが，SCSI接続タイプのPCカードスロットなら特にドライバを必要とせずにフラッシュメモリカードなどをアクセスできることも確認されています。さらにHDDの1Gバイト制限を超える方法などが発表されました。これらのSCSIソフトに関する技術蓄積は，X68000の外部記憶大容量化への福音となっています。

ただ，現状でも相当数いると思われる10MHz機ユーザー(お下がりや中古購入なども比較的多い)が使えるSCSIボードがないといった，まだまだCD-ROMドライブが誰でも備えている周辺機器といえない部分も残されています。近いうちに予定されるMach2pの開発により解消することを目指しています。

●ハードウェアの高速化

ハードウェア(特にCPU)の高速化に関しては，060turbo(1997)をリリースしたこと，これからリリースされるJupiter-Xにより，そこそこ快適な速度を持った環境を提供してきました。ただ，これも製品として出荷するための技術蓄積を伴うもので，満開製作所がこうした製品を量産するまでには先達の智恵を学ぶところから，実務的な部分の習熟など，相当の期間がかかってしまったのも確かなことです。特にハードの設計は外部の方のものですが，060turbo.sysなどのドライバ，アセンブラなどの060対応などは，ほぼ完全に内製の形でリリースしてきました(詳しくは060turbo，68060に関する記事を参照)。アートワークなどもかなりの製品で内部で行うようになっています。

●CD-ROMソフトへの展開

また，1996年頭の時期から，CD-ROMディスクマガジンのリリースを積極的に行ってきました。1997年には，当初難しいと思われた定期刊にも着手し，現在も「激光電脳倶楽部」として順調に刊行を続けています。CD-ROMでなければ実現できないことを主眼に，雑誌としての体裁を整えながら，現在もさまざまな試みを行っています。

CGA作品をCDからダイレクトにX68000でムービーとして全編再生する企画や，広いマップを使ったゲームや，RPGインタプリタの公開，CDの音声トラックを使ったMIDI曲，ボーカル曲の掲載，NetBSD1.3(非公式版)の掲載，多数の画像を使った読みものなど，CD-ROMの容量を毎号フルに使ったものになっています。これは見てもらうのがいちばんですが，現状のスペックでも，かなりのものを作り出すことができるのだ，ということを物語っています。また，X68000界におけるさまざまな素材・ツールの提供の場，そしてユーザーの作品発表の場となっています。

●インターネットへの接続

WWWについては，電脳倶楽部別冊16号以降，積極的に取り上げており，ひととおりソフトウェアを揃え，ダイヤルアップ接続することに関してはX68000でお金をかけずにできる環境が作られています。

また，最近は月刊の磁性面上で，ネットワークに関する解説記事も連載されており，ネットワークプログラミングにも踏み込んだものになっています。やはり，単純にアクセスして利用するだけではX68000らしさが足りません。ネットワークプログラミングこそ，新しい面白さを発見する場となるでしょう。

またグラフィックを扱えるWWWブラウザも内製，公開されており，X68000でのブラウザのデファクトスタンダード(事実上の標準)を狙っています(WebXpression)。

今後，この分野からは目が離せないといえるでしょう。

●ネットワークへの対応

OSがネットワークに対応していない，事実上イーサネットボードがない，というのが現状としてはいちばん立ち遅れた分野でしょう。Human68kには，バックグラウンド機能がありますが，Humanの内部バグもあり，ほとんど生かされていない状況です。今後OSの公開と積極的なバージョンアップを行う必要がある，といえるでしょう。

そして次世代機へ

最終的には，やはり満開製作所が「NewX」に相当するマシンを作らなければならないでしょう。やらなければならないし，やるのは大前提でもあります。ただ，そのためにはハード/ソフトとも，さらにノウハウを蓄積，研究していかなければならないというのが現状です。取り急ぎ，骨子ができましたので(満開製作所からのお知らせ)，それもあわせてお読みください。

最後に，パーソナルコンピュータシステム全般において，我々は海外からやってくるものを餌を待つ雛のようによだれを垂らして待っているつもりはありません。だって，電子立国でしょ？　日本は。

満開製作所が「X68000最後の砦」とユーザーの方々からいわれるのは大変ありがたいことです。ユーザーがいてこそ守る砦の意味もあるわけです。ユーザー自身が現在置かれた状況を積極的に切り開いてくれることをスタッフ一同，心から願っています。

同人ハードとはなにか？

今回の原稿のなかにはなにげなく「同人ハード」という言葉が使われているが，X68000関係に詳しい方ばかりでもないだろうから，いったいどんなものなのだろうかと不思議に思っている人もいるだろう。

「同人ソフトのハードウェア版」だといってしまえばそれまでなのだが，まさに個人で設計して作った拡張ハードウェアの類のことである。ジャンルは実にさまざまだ。連射装置とか，高音質ケーブルとか，プレステのメモリアダプタとか，X68000で有名どころではカラーイメージユニットやLAN関係のNeptuneなどがそういったものに入るだろうか。

また「なんで，VenusとかJupiterとかNeptuneとかま～きゅりーなんだ？」と思う方がいても至極当然だろう。これはMercury Unitに端を発している。すでに伝説化しているエピソードだが，かつてAD PCM(圧縮PCM)しか持たないX68000で高音質な久川○の声を聞きたいという，それだけの目的で製作されたのが元祖Mercury Unitだ。高音質な非圧縮PCMデータはほっておくとすぐにハードディスクを埋め尽くしてしまう。週末ごとに膨れ上がるデータに，どう対処すればよいのかというと，作者いわく「DATストリーマに入れて管理しているので問題ない」……という非常にロジカルな回答が得られている。一般にPCMボードといったときに要求されるであろう機能をあっさり省いた潔さが成功の鍵だろうか。結果「AWE64GOLDって音悪くて……」というようなユーザーに愛用されるようになる。

以来，Jupiter，Neptune，Venus……など，全然別の作者の大作が続いている。ほかにも関連する名前のものはたくさん作られており，広く知られていないものをあわせると膨大になると思われるが，メジャーネームには超アマチュア級の作品だけが残っている感じになっている。

満開製作所からのお知らせ

X680x0互換機の製作を公式に発表 (1998.08.18)

概要

満開製作所では、西暦2000年中の発売開始を目標として、X680x0互換機を製作する計画(計画名「零式」)を立案中です。また、この零式計画のために、1999年中に先行的に試作機を製作し、零式の礎とするための補助計画(計画名「九九式」)を企画しています。

まだ未定の部分も多いのですが、今回はこれらの計画の骨子を公式に発表します。

基本コンセプト

●いたずらにスペックを追求しない

こういった計画では、「性能はどうなるのか」「**を載せてはどうか」といった、スペック偏重なことがらが第一の関心事になりがちです。しかし、それは手段を論じているだけにすぎません。

そうではなく、シャープが撤退したいまとなっても熱心なユーザーが数多く活動し続けているX680x0の現状を踏まえたうえで、我々はどのようなものを作っていくべきなのか、いまあえて開発を行う意義と目的を考えました。

そのような意味で、スペックは副次的なものです。コンセプトを実現し、さらにユーザーの皆さんに受け入れられるものを目標として策定されることになります。

●存在意義：「開発にあたって支障のないマシンを」

現在のパーソナルコンピューティング環境は、Macといった存在もありますが、事実上Windows一色です。そのような存在に張りあってもしかたありません。

ところで、現在のWindowsは、道具として見れば、堅牢ではないということを差し引いても、ソフトウェア資産の面で非常に優秀ではありますが、構造的しがらみも多く、コンピューティングの勉強にはまったく向いていません。

では、X680x0では支障がなかったかというと、ユーザーに対する情報が一貫していなかったと認めざるをえない部分もあります。そこで、開発にあたっての敷居が低いマシンというのを、重要な目標として掲げます。X680x0は、我々にとっていわば「学校」のような存在ですが、我々はさらにその路線を突き詰めていき、それをひとつの「魅力」にまで高めていきます。例を挙げるなら、「ゲームを作りやすいマシン」ということになると思いますが、もちろんユーザーさん次第で「ゲーム」の部分はほかの単語に置き換わってもよいでしょう。

●互換性は重視しない

100%の互換性を確保するのはあまりに莫大な労力が必要になりますし、ソースがあれば新たにリコンパイルするなり手を加えるなりすれば、より大きなパフォーマンスを得られる可能性があります。だとすれば、完全な互換性というものがあまり重要とは考えられません。

また、新しい機能に対するブレーキともなりかねません。新しいことを行うのに、古いことがしがらみになることの愚かさ、苦しさについては、過去に反面教師が多数存在しています。

スプライトやグラフィックについても、それらの機構を丸ごと互換性を堅持しつつ実現することは、かなりの手間と費用を要します。そこで、拡張性を残しつつ、従来の動作はエミュレーションで再現することを検討中であり、技術的な可能性を九九式で探っていきます。

●OSはHumanの実績を踏まえつつ、独自に開発

Human68kの実績を踏まえつつ、使いにくいところは徹底的に改良した、独自のOSを標準添付させる予定です(詳細は未定です)。

また、オプションとなりますが、NetBSDなどを動作させるのに支障が出るようなことは極力避けます。

●必要な情報はすべて公開

OSやIOCSに該当する部分のコールだけでなく、必要と思われる部分はすべて公開します。OSのカーネル部分なども例外ではありません。これには、シャープやハドソンなどの権利に抵触しないよう、独自に開発したことを示す意味もあります。また、ROM内ルーチンや回路図、FPGAの内部論理なども、可能な限り公表します。

●開発に必要なものはすべて標準添付

フリーウェアのなかには販売を禁止しているものもあり、どこまで実現可能かわかりませんが、少なくともフリーウェアを収めたCD-ROMを無料で添付するなど、開発に必要なものはすべて揃っているという状態を理想としていきます。

今後の展開については、具体的な進展があれば本誌、満開ネット、ならびに弊社ホームページ上で公表していきます。　(文責：中村隆生)

表1　基本的なスペック

以下はあくまで予定であり、今後の技術の進展により変化する可能性が非常に大きいことを念頭に置いてください。

- **MPU**：MC68060(50MHz)
  サブMPUまたはDSPの搭載を予定
  PCIバスブリッジ直結
- **メインメモリ**：Synchronous DRAMにより構成　理論上限は約4Gバイト(予定)
- **グラフィック**：高性能グラフィック用チップで構成、PCIバスまたはAGPバスで接続
- **I/O**：
  - FDドライブを廃止(代替手段を策定)
  - HDD、CD-ROMドライブを標準装備
  - Ethernetポート(スペック未定)
  - USBポート(予定)
  - SCSI-3ポート
  - 高速(150kbps)RSポート
  - IEEE1394ポート(予定)
- **添付物**：OSおよび関連ソフトウェア、マニュアルセットおよびオーナー用CD-ROM、キーボード、マウスほか
- **その他**：未定、策定中

いかにして満開製作所はCD-ROMを作るようになったか

船本昇竜/あいはらてつや

すてきな電脳倶楽部

すてきな電脳倶楽部，通称「すて電」。長い準備期間(説得期間ともいう)を経て「とにかく試しに一度だけ，CD-ROM媒体の電脳倶楽部を作ってみよう」と弊社代表取締役祝一平にいわせることに成功。記念すべき満開製作所製X68k用CD-ROM第1弾として制作，1996年2月発売されました。

純粋に「CD-ROMを媒体とした電脳倶楽部」を目指し，開発コードを「92.5号」にし，容量制限がないことを除けば，制作方法もFD版電脳倶楽部とさほど変わらないものでした。

とはいっても，さすがに容量が2桁違うというのは相当なもので，画像，CGAPICデータをはじめとする，FDでは収録すら不可能な巨大なデータをいくつも収録し，それらを簡単に閲覧/実行ができるように……。また，これまでは文字だけメニューにも，ちょっとしたイラストカットがつき，なにげない部分もより華やかなものとなりました。

パッケージもなかなかのものでプラスチックケースに映える岡村画伯描き下ろしの「すてきな」ジャケットイラスト。さらにCDレーベルにも描き下ろしSDイラスト。PSやSSのソフトと一緒にCDラックに入れても，バッチリきまる背表紙。

Oh!X休刊と入れ替わるように登場し，それもメーカーが発売/サポートしていないメディアで，それどころか，すでにメーカーが手をひいたマシンを対象にしているなど，もはや市場には不安要素しかありませんでした。そんな周囲の不安をよそに，初回生産の千本は予約完売。日経産業新聞でいうところの「CD-ROMのヒット本数」である3千本も4カ月目に達成。現在までに，約5千ものユーザーの手に渡る名実ともに大ヒットとなりました。

裏話などについては，もう，随分昔のことなので，漠然と大変だったということしか覚えていません(←ウソ)。それでもひとつ挙げろといわれれば，機材にCD-Rがなかったのは，少々辛かったような気もします。HDやMOを仮想CD-ROMに見立て作業を進めるのはいいのですが，最終段階近くなってから，コチョコチョとした仕様の違いや，CD-ROMデバドラのバグなどには，正直，勘弁してほしいほど悩まされました。

また，当時の編集部の標準MOは128Mバイトだったので，すて電の中身は128M，というウソだか本当だかわからない話もあります。のちに，ゲームプログラマの友人に，「ROMライタなしで，スーファミのソフト開発するくらい無茶やで」と激しい関西弁のツッコミを入れられました。　(F)

すごい電脳倶楽部

すて電の予想以上の反響のよさと「次のCD-ROMにも期待しています」との声援に励まされ，月刊電脳倶楽部100号記念までに間に合わせるべく「次」の制作準備を開始したのでした。

当初の企画では，「原稿のFD別冊をたたみ，CD-ROMで『季刊すてきな電脳倶楽部』を新創刊する」つもりだったのが，まぁ，実にモロモロの複雑な理由で企画は倒れ，タダで倒れるのはクヤシイので，単発CD-ROM版電脳倶楽部第2弾，通称「すご電」として1996年8月に発売しました。こちらも，現在までに，4千以上のユーザーの手に渡るヒットになっています。

コード名「すて電2」は，CD-R(当時，私物)を導入することで，「すて電でできなかった/弱かったことを実現/強化」，「CD-ROMならではの色を強く」をキーワードに進行し，オーディオトラックが扱えるようになり，MOの容量にとらわれない編集もできるようになることで，企画の幅は広がりました。オーディオトラックへのMIDI曲収録はもちろん，CGAコンテストグランプリ作品の画像収録，デジカメのサンプル画像付きレポートなど。従来のコーナーもさらにパワーアップし，岡村画伯の描き下ろし4色スーパースクリーンピクチャーレーベルなどのおまけもありました。デュプリ会社の話では，おそらくパソコンソフトでは初めてのケースとのこと。

それにしても，2本目のCD-ROMであることも手伝ってか，「すご電」として発売するということが決まってからは，脳裏にこびりつくような思い出や記憶の類がほとんどありません。　(F)

激光電脳倶楽部

激光電脳倶楽部。略してゲキデン。アニメのゲキなんとかガーとは全然関係ないのですが，よく考えると，時期はそう遠くないですね。鶴亀。

時に1997年初頭。満開製作所の体制はボロボロになっていました。

そのようななか，FD版別冊(季刊：現在廃刊)ではどうにもならない，というかなり切羽詰まったところで企画され，そしてGOサインが出たのが，定期刊のCD-ROMでした。

すごい電脳倶楽部から約1年。遠回りはしたものの，6月末(正確には7月上旬)，晴れてこの時期に出荷開始された060turboとともに，新しい

製品ラインアップとして現在に至る，というわけです。

さて，激電ではそれまでの2回のCD-ROM発行により，制作する側にも多少の余裕ができたといえますが，それまで以上に大容量化を意識したものになり，やれることはすべてやる，というスタンスの雑誌になりました。

これまで投稿雑誌としてのスタンスを崩さなかった部分も，多少変えつつ(もちろん投稿が最重要なことは変わりませんが)，内容的にも，見て聴いて楽しめる部分を拡張してきました。また，編集者1人ひとりがやりたいことをドーンと取り上げる，という比較的個人の資質による構成になってきました。やりたいことがストレートにできるというのは，それだけで力になるものです。

それがよかったか悪かったかについては，まだまだ予断を許さないところもあります。当然ながら，CD-ROMによって解消された制約があった半面，よりX68000の制約を強調することになってしまった面もあります。

ムービーにしても，マシンパワーにあわせてデコードのいらない(逆にCD-ROMドライブのパフォーマンスの向上にあわせた)フォーマットでなくてはならなかったことや，大規模化することによる必要メモリ量の増加などもあります。

内容に関しては，これまでのCD-ROMの目次メニューをCD-ROMに掲載させてもらいましたので，ぜひそちらを見てください。HTMLでも収録してありますので(DSHELL[*1]ドキュメントをイラスト込みでHTML化するのもperlなどを使えば比較的楽にできるようです)，他機種をお持ちの方も見てほしいところです。

現状では別のプラットフォームを併用しているユーザーも多くなり，その点では，HTMLはどうかとか，Javaはどうするのか，などなど，配慮の必要な時期にきているといえるでしょう。

すでに，CD-Rドライブが普及してきたこともあり，コミケでもCD-RでディスクマガジンをI制作するサークルも増えています。そろそろ新しいスタイルのものが出てきてもいい頃かなと踏んでいますが，激電を凌駕するようなものがいつ出てくるか，非常に楽しみですね。 (A)

やさしくしてね

図1

＊1 DSHELL：電脳倶楽部のシェルでありランチャ。ブラウザといった方が現在はしっくりくる人が多いでしょう。独特のマウス操作体系を持っています。

総合パターンエディタ「てぺ」について

たんぼ(TNB製作所)

「てぺ」を使う利点・意味，それは作者の制作経緯のとおり，ゲーム制作にあるといえます。ここでは，それを踏まえて「てぺ」の特徴を紹介しましょう。

「てぺ」はパターンエディタ(ゲーム画面をチョコチョコと動くキャラクターを描くもの)であり，なおかつグラフィックエディタ(アドベンチャーゲームで例えると，人物のCGや背景を描くもの)です。ですから，ゲームで使用されるビジュアル部分の制作のほぼ全域をサポートすることができます(3DCGなどは無理ですが)。

ではまず，RPGを作るということを考えて，ビジュアル部分を制作するうえでの「てぺ」の効果を示していきます。

・マップ上を歩き回るキャラクターやそのマップ用の背景BGパターンを描く

ここではキャラクターの歩いたりするアニメーションを確認する機能が役に立ちます。32×32～256×256(ドット)のサイズのアニメーションを最大8パターンまで確認でき，描いている画面の位置を指定することでパターンを指定できますので，バリエーションを作るのも非常に簡単になっています。

・モンスターのグラフィックを描く

ドラクエやウィザードリィ形式のものを想定します。大きい画像だったりすると，なかなかマウスだけでは描けないものです。「てぺ」はさまざまな画像形式に対応できますので，スキャナで取り込んだ画像ファイルを読み込むこともできます。また，描画機能としてマスクがあるので，込み入ったグラフィックなども制作できるでしょう。パターンエディタではありますが，240色モードがあったり，256色パレットを16バンクまで持てたり，512×512の画面をフルに使って描くことができ，多様な画像フォーマット(PIC，PT4，CUTなど)で保存することができます。

・マップを作る

草原や山，城・町などを配置していくためには，マップエディタを使います。ほかにもマップクリップという(いわゆるクリップボード)ウィンドウがあり，BGとしてバラバラに登録されたパターンを，城なら城の形に組みあわせて置いておき，マップエディタ上でいつでも使うことができます。たとえば，頻繁に使う色違いや大きなパターンを使用するときに便利です。

ほかにも，フォントのエディットなども可能です。12，16，24ドットのフォントが作成できるので，ゲーム以外でも重宝しそうです。

このように，ゲーム制作に必要と思われるさまざまな機能がたくさんあって，逆になかなかとっつき難い部分もあるかもしれませんが，自分の必要とする部分だけを使用していくので構わないと思います。興味を持たれた方はぜひ使って，なにかを創ってみてください。

てぺを使う人

鈴木 浩

「T・総合パターンエディタ(T・Total Pattern Editor)」の略称TTPE→T2PEの愛称が「てぺ」です。「てぺ」はTNB製作所(同人)が著作権を保持する「FSW」です。

「T2PE」と初めてタイプしたときになぜか「てぺ」と読んでいたのです。総合といっていますが，それほどたいしたエディタではありません。なにが総合のエディタなのでしょうか。

昔，DQのようなタイプのゲームを作っていたときのこと。スプライトなどのパターンはHumming Bird Softの「SPRITE EDITOR PRO-68K Terazzo」を使って描いていました。このソフトはスプライトやBG，つまり8×8や16×16ぐらいの大きさのパターンを描くことを前提に設計されたものでした。パターンのアニメーションパターンを確認したり，すべての環境をセーブし，後日その続きから簡単に始められる，という凄く便利な機能がいろいろついていました。が，マップエディタはお世辞にも使えるものではありませんでした。そのとき唯一使えたX-BASICを使って自分用のマップエディタを作りました(確かこれがX68000で初めてのプログラム)。我が子は可愛いもので，人に見せられないものでも当時は，それとなく納得できるものでした。

道具がなんとか用意でき，やっとゲーム制作に入りました。しかしパターンを描いてセーブしては終了，マップエディタを起動してパターンを並べる，あ，このパターンが足りない，ずれてる！ セーブしてマップエディタを終了，またスプライトエディタを起動，ロード，修正。しかも当時HDDも持っておらず，それぞれのソフトも別々のFDに分かれていて，毎回交換しないといけないという手間があったことも忘れられません。

「面倒」と思いながらも「しかたがない」とそのゲームを完成させたあと，次回作に向けてまず思ったのが「パターンエディタとマップエディタをくっつけたものがほしい」ということでした。しかし，そんなものはありません。ないものは作る。ゲームを作ると同時に覚えていったCを使い，その制作を開始しました。それが「てぺ」です。

X-BASICで作ったマップエディタと「Terazzo」を参考に，構想を練り，制作していきました。日記によると制作開始は1993年の3月のようです。ちなみに，てぺ制作日記を残していまして，読み返すと自分でも面白いものです。

現在，当初の目的(ゲームの次回作)は達成されていないまま，僕は満開に籍を置くようになり「てぺ」は「240色グラフィックエディタ」＋「マップエディタ」＋「簡易テキストエディタ」＋「α」，そして「ドット単位のアンドゥ」「ウィンドウシステム」という，ある意味で総合された，巨大ツールになっています。

僕自身，月刊電脳倶楽部「我夢我行」連載内で，2値のCUTファイル(画像)を多用していることもあり，「てぺ」には2値画像を描くための機能が充実してきたことも付け加えておきます。

X680x0用ハードウェアのこれまで，現況，そして将来

中村隆生

ここでは，X680x0用のハードウェアについて，休刊後から現在（98年7月末）に至るまでの経緯と，現在の状況，そして将来の構想を，分野別に紹介します。

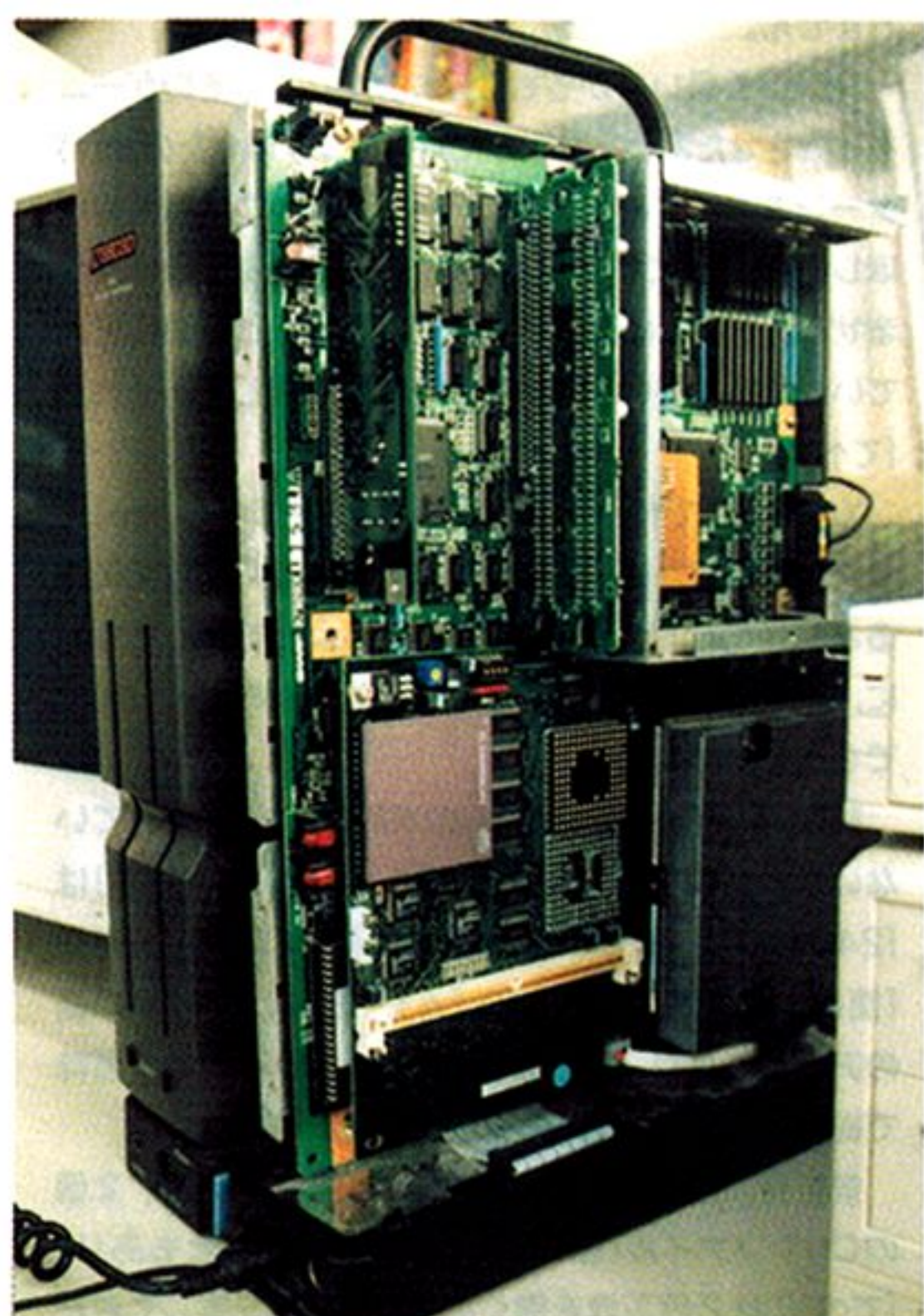
CPUアクセラレータ

X68030用のアクセラレータは，BEEPs氏による040turboを計測技研が量産していましたが，ずいぶん前に生産完了となりました。これはMC68030を引っこ抜いて，代わりに差すタイプのアクセラレータで，この方式に先鞭をつけた偉大なボードといえるでしょう。

●060turbo

その040turboの成果を活かして，満開製作所で060turboを発売しました。こちらは現在も入手可能です。MC68060を50MHzで駆動し，Dhrystoneベンチマークでは75,000を叩き出しています。128MバイトまでのSIMMが装着可能で，060内蔵のMMUと付属のドライバにより，128Mバイトをメインメモリの一部のように利用することができます。動画取り込みなどのメモリ喰いな分野では，処理の速さとあいまって，絶大な威力を発揮するでしょう。

従来ソフトとの互換性は040turboと同等です。つまり，自己書き換えをやっていたり，ハードウェアを直接叩いたりしていなければ，十分動作します。また，040turbo同様，コピーバックモードとライトスルーモードがありますが，さらに060turbo独特の機能として，スーパースカラとブランチキャッシュが切り換え可能です。

ただし68060からはmovep命令がなくなりましたので，ソフトウェアエミュレートで対処しています。ですので，movep命令を連発しているようなソフトでは，パフォーマンスが落ちることがあります。

060turbo搭載のX68030下側に見えるSIMMソケットにはローカルメモリが搭載可能だ

●Venus-X

この分野で特筆すべきなのは，X68030用の同人ハードウェアVenus-X（まさちく工房）でしょう。これはMC68030を倍速で駆動し，さらに2次キャッシュを搭載するという，同人の域を超えたハードです。

MPUの倍速動作だけでは，（HARPが示したように）たいしたパフォーマンスは出ないのですが，2次キャッシュを搭載したことにより，2倍以上の動作速度をコンスタントに叩き出し，新しい可能性を見せてくれました。なお，頒布はすでに終了しています。

●Xellent30

X68000用のアクセラレータとしては，Xellent30シリーズがありました。このシリーズは，旧機種に68030を載せてしまうというもので，そこそこのパフォーンマンスとそこそこの価格で，しかもACE，EXPERT，PRO，XVIと対象機種が広く，かなり人気を博しましたが，現在では完全に生産完了になっています。このため，現在新品で入手可能なX68000用アクセラレータはなく，満開版Jupiter-Xを待つしかありません。

●Jupiter-X

Jupiter-Xは，68000機種に68040/060を載せてしまおうという大掛かりなアクセラレータです。040turboなどとは異なり，ローカルメモリを持っているので非常に高速な動作が可能です。β版までは68040/68060搭載という計画でしたが，現在デバッグ中のいわゆるγ版では68060専用になっています。これを50MHzで駆動し，060turboと同等かそれ以上のスペックが期待できます。また，SIMMソケットを3スロット装備しており，使い方の幅が大きく広がるのも嬉しいところです。対象機種はX68000 ACE，EXPERT，SUPER，XVIです。ただし，Jupiter-Xの取り付けにあたっては，X68000本体の改造が必要です。これについては，取り付けサービスの実施も計画中です。

メモリ

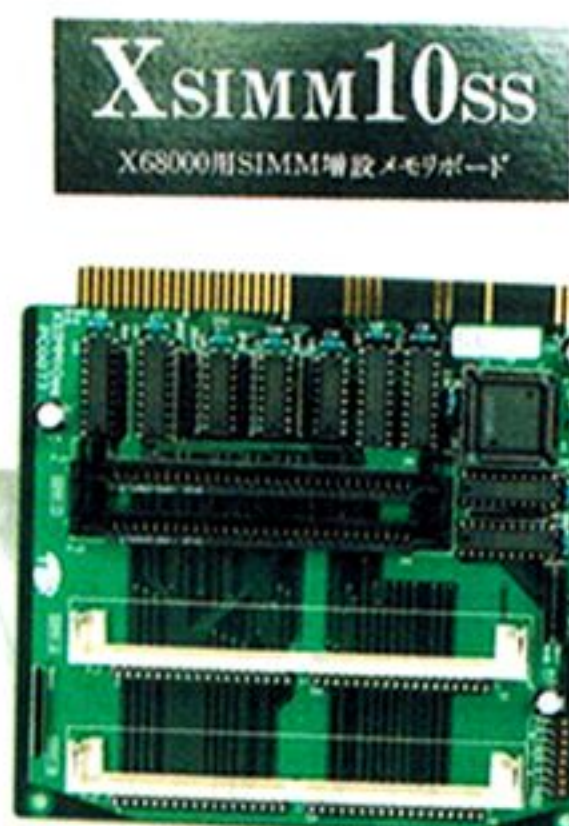
●XSIMM

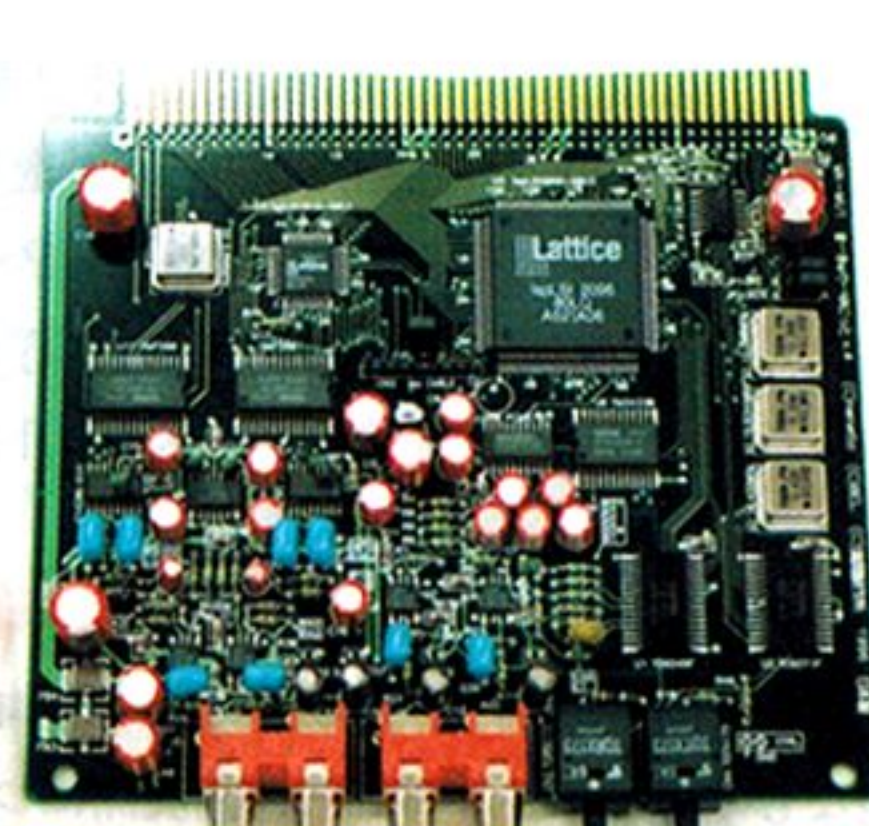
X68000用では東京システムリサーチからXSIMM VI，XSIMM VIc，XSIMM 10ssが出ており現在も入手可能です。ただし，XSIMM VI/VIcには4M×16個構成のSIMMを使用する必要がありますが，このタイプのSIMMは中古以外ではもう手に入らないと考えて間違いないでしょう。

これに応える形で，満開でMKS6M60Nという専用の6MバイトSIMMを内製しましたが，DRAMがすでに生産完了になっているため，販売価格はかなり高価です。ただしXSIMMVI/VIcでの動作を完全に保証しています。

X68030用であれば，満開のMK-5BE8が在庫

XVI専用として登場したXSIMM Ⅵ。SIMMが使えるというのは画期的だった

CompactXVIの親ガメになるXSIMM VIC

ノーマル機用拡張スロットで一気にフルメモリになるXSIMM10SS

こちらは満開製のX68030用増設メモリカード

これが満開版のま～きゅりーゆにっとだ。FM音源を搭載している

こちらは光出力端子。高音質を求めるならこれを使うしかない

ありの状態です。これを装着すれば即フル実装になります。なお，X68030はデフォルトではスタティックカラムモードがイネーブルになっていますが，このモードのチップが入手困難であったため，ボード上のPLDで疑似的にスタティックカラムモードを作り出しています。

各種ボード

68業界初のバスマスタ転送SCSI-2ボード「Mach-2」は，部品の入手難から製造完了となりました。サードパーティ製のSCSIボードも全滅ですから，現在新品でのSCSIボードの入手は不可能です。ツクモからは，SCSI-1ではあるものの，パラレルポートとSIMMソケットを備えた，TS-6BS1mkIIIという製品が出ていましたが，こちらも生産完了で新規入手は不可能です。

ツクモといえば，後期にはかなり面白い製品を出していました。特にTS-6GA1という拡張スロット用のグラフィックアクセラレータボードが挙げられます。これはAT互換機用のCRTCを拡張スロット経由でボード化したものですが，グラフィックアクセラレータというよりは，高解像度対応のフレームバッファというべきものでした。ただ，このCRTCが入手困難になり，100枚ほど製造したところで生産完了になったということです。可能性の広がる面白いボードだったのですが……。

●Neptune-X

同人ハードとして，AT互換機用のイーサネットカードを68バス用に読み替えるボード「Neptune」が，いくつかのサークルで製造・頒布され，好評を博しました。

これは基本的にバス読み替えボードであるため，TTLで作れてしまうことと，回路図が原設計者のShi-MAD氏の手によりホームページ上で公開されたこともあり，自分で作ったというユーザーも少なくないようです。

●PSX16550

また，草の根ネット「PowerSTATION」においては，高速RS-232Cボード「PSX16550」および「PSX16750」が開発され，頒布されました。従来のX68000では不可能だった115kbpsを実現しており，BBSホストを運営している人や，ヘビーな通信ユーザーの間で大人気でした。なお，現在頒布は行われていません。

●ま～きゅりーゆにっと

その他のボードで入手可能なものといえば，満開のPCMボード「まーきゅりーゆにっと」があります。もともと草の根ネット「トワカルト」で開発されたものを，満開で製品化した形です。32/44.1/48kHzのステレオ16ビットPCM音声をデジタルのまま取り扱うボードで，光入出力端子を備えています。また，音源付きバージョンはOPN音源LSIを2個搭載しており，高度な演奏が可能です。

周辺機器

GORRY氏の優秀なフリーウェアsusie.xのおかげで，CD-RとCD-RWドライブ以外は極端なハードでなければ，市販のSCSI機器が十分利用可能です。また，アイ・オー・データ機器の高密度3.5インチドライブFDS-120は，たんぼ氏のフリーウェアfds120p.xと先述のsusie.xを併用することで利用可能です。

X68000のSCSIについて

たんぼ(TNB製作所)

Oh!Xが休刊して3年近く。3年前というと満開製作所からX68000用SCSI-2ボード，Mach-2が発売された頃。開発中のMach-2を見せてもらったのがすべての始まりだったかもしれません。

あ，満開のたんぼ(TNB製作所)です(カッコ内までペンネーム)。

Mach-2が発売されて以降，X68000のSCSIハード環境自体には変わったことはひとつもありません。世の中のSCSI機器は高速に，かつ大容量へと変わっていきましたが，SCSI-1の頃に発売されたX68000の純正，あるいは互換ボード(Mach2以外のSCSIボード)でも，容易に新しいSCSI装置を使用することができました。

ただ，大容量化による問題が露見しました。「1Gバイトを超えたHDDはX68000で認識できない」というものです。調べてみると「最後のパーティションの先頭が1Gバイト未満でないと認識しない」ということが明らかになりました。

たとえば2GバイトHDDでパーティションを2つ作成する場合，先頭のパーティションのサイズは1Gバイト未満でなければならなかったのです。そのため，しばらく「X68000でSCSI HDDを使う場合には，1Gバイト以下の機種でなければいけない」という話になっていました。

が，これは調べてみると，FORMAT.XによってHDDに書き込まれるIPLやデバイスドライバにミスがあったのでした。SCSI規格では，扱える範囲(大きさや長さ)が違う読み込み命令が複数あります。X68000のIPLは，扱える範囲の小さい読み込み命令だけを使用してパーティション情報を得ようとしていました。ですから1Gバイトを超えるところが読み込めず「エラー」となり，結果「アンフォーマット」のHDDとなってしまい認識されていなかったのです。

解決法は簡単です。FORMAT.xによって書き込まれたHDD上のIPLやデバイスドライバを取り替えるだけでいいのです。

というわけで，現在，1Gバイト以上のサイズのHDDも難なくX68000でフォーマット，使用，起動ができるようになっています。

X68000のSCSIで変わったことといえば，主にソフトウェアによるものです。Mach-2を差したX68000の，元からあった内蔵のSCSIポートも共有しX68000 1台にSCSI機器が14台接続できるようになったことも書いておきたいところです。

また，多様化するSCSI装置のうち，シャープのJXスキャナシリーズ，アイ・オー・データ機器の3.5インチFDD FDS-120，またまだ完全ではないですがDVD-RAMドライブもX68000に繋ぐことができるようになっています。ついでにX68000同士をSCSIで繋ぎ直接ファイルやデータのやりとりをすることも行われています。

「SCSI」という規格はSCSI-3になり，シリアル化が進んでいくと思われますが，従来のパラレルタイプもしばらくは現役で残っていくでしょう。それらは，今後もソフト次第で十分X68000で使うことができるものです。これは，X68000の「SCSIが素直だから」というよりも，自由にSCSIに触れるから，どうにでも対応できる，ということなのです。

●スキャナ
スキャナについては，満開のSTOOL.Xを使用すれば，JX-250W/JX-350Wといったシャープ製のSCSIスキャナが利用可能です。

アクセサリ

同人ハードでは，CZ-6VT1（イメージユニット）互換の高画質動画取り込みユニットがいくつか製作されています。コミケで購入可能なものもありますので，要チェックです。コミケといえば，まだまだ数多くのX68000の同人ソフト/ハードのサークルが出店しています。

また，キーボード変換装置もいくつか発表されています。なお，AT互換機とX68000の信号を変換するXSel68は，ソフトウェアの開発が遅れており，まだ頒布に至っていないようです。受け付け自体は終了しています。

マウスに関しては，満開のMK-MJ2が発売されています。これはPS/2マウスの信号をX68000用に読み替える装置です。3段階の速度切り換え機能がありますが，これはMK-MJ2側でデータを加減しているのではなく，PS/2マウスにもともとそういう速度調整機能が備わっているからです。ですので，手抜きPS/2マウスでは，もしかするとこの速度調整機能が使えないかもしれません。なお，しばらくは生産が継続される予定であり，入手可能です。

発売が比較的近い製品

ここでは主に，満開製品に限ってですが，発売が近い製品についてざっと触れておきます。

●4倍速スキャンコンバータ
まず4倍速スキャンコンバータMV-SC1がデバッグ中です。これは，S端子・ビデオ端子・21ピンEIAJ端子からのビデオ信号や，15ピン端子からのX68000の映像信号を，走査線をコピーする手法により倍速・4倍速スキャンして，31/62kHzで映し出すための装置です。結果的に，上記の入力信号を，15kHz信号に対応していない最近のマルチスキャンモニタで見ることができるものです。フィールドメモリを使用することにより，動き適応フィールドスキャンも可能となっています。

たとえば，ビデオデッキとLDとゲーム機とX68000，という4系統を切り換えて，ひとつのマルチスキャンモニタに映し出すということが可能です。入力切り換えは，X68000のキーボードからもコントロール可能です。もちろんX68000がなくても利用可能です。

●Jupiter-X & Mach2P
次に，Jupiter-Xがデバッグ中です。

また，SCSI-2を中心に据えた複合ボードmach2pが企画進行中です。大好評であったMach-2を核に，PSX16550相当の高速RSポート，増設メモリ，予定ですがイーサネットポートを備えて，スロット不足に応えるものです。

今後の満開製作所のハードウェア展開

ここからは，具体的な企画ではないものも含み，将来の製品展開について触れておきます。

まず，X68030用の4スロット拡張ユニットが挙げられます。東京システムリサーチがXpander IVとしてX68000用の4スロット拡張ユニットを発売し，好評だったのですが（現在は生産完了），これの030版は製品化されないようなので，満開でやってしまおうというものです。ただし，時期はまったく未定です。

次に，PCIブリッジ計画があります。68の拡張スロットはスループットからいって，バスマスタ技術を利用したところで限界が見えていますから，MPU直結のPCIバスブリッジを経由してPCIボードを利用できるようにしようというものです。技術的なメドはついており，あとは人的と時期的な問題を残すのみとなっています。

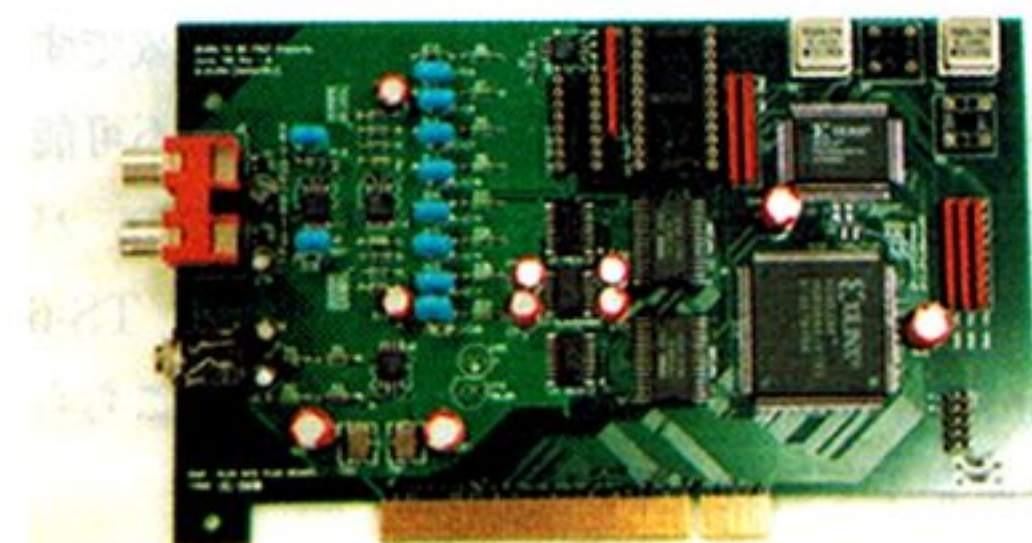
これが実現すれば，TS-6GA1の後継グラフィックアクセラレータが安価で実現できるのでは，と踏んでいます。問題は，市販のPCIグラフィックカードのデータシートが手に入るかどうか？というところなのですが，これはなんとかなりそうな感じです。

以前ツクモから出ていたハイメモリ関連についても，製品化の方向で検討中です。特に，最近はDRAMとSDRAM，DIMMとSIMMの価格が急速に接近していますから，SDRAMのDIMMを利用できるようにしようという企画もあります。

また，Venus-Xの技術を応用した，68060用の2次キャッシュユニットも検討しています。どのような製品化を行うかはまだ未定ですが，このとき同時にPCIバスブリッジとハイメモリも搭載してしまい，040turbo/060turbo/Jupiter-X用のグレードアップキットとして出すようなかたちで漠然と構想しています。

3.5インチ外付けFDドライブについては，FDS-120のドライバ完成により，一応決着を見たかたちですが，起動可能なFDドライブを製作する予定もあります。ただし，オートイジェクトドライブはすでに入手不可能なため，手動イジェクトとなるでしょう。

現在では上記に書いたものを含め，少なくとも6つのプロジェクトが動いています。すべてモノになるとは限りませんが……。

最大の目標としてはX680x0互換機が挙げられます。これについては別項を御覧ください。

これはちょっと変わった同人ハードのFM音源カード。PCIバスになっている

「マウスジャック2（MK-MJ2）」について

満開製作所　小南淳一

以前満開製作所では，PC-9801用マウスをX68000に繋ぐアダプタである初代「マウスジャック」を販売していましたが，これは製品内部の変換ICの入手が不可能となり製造できなくなってしまいました。それに対し，新製品「マウスジャック2」はいま流行のPIC16F84をCPUとして使用した，オリジナルの製品です。「マウスジャック2」では98マウスの代わりにPS/2マウスを使用します。現状では，こちらのほうが入手が容易で製品の種類も多いなどの点でベターな選択といえるでしょう。

マウスの速度切り替え機能も新たに装備しています。「マウスジャック2」のケース内にあるジャンパを差し替えることで速度が3段階に変わります。切り替え作業のためにケースの蓋を開ける必要があるのは賛否両論なのですが，いったん決めてしまえば頻繁に速度を切り替えることはないだろうとの判断から，現在の形になりました。切り替えについてはソフトウェアで処理する方法もありますが，ポインタが2ドットずつ動く不具合が出る場合があります。そのため「マウスジャック2」ではハードウェア的に対処しています。実はPS/2マウスは標準で速度切り替え機能を持っているので，これを使いマウスレベルで速度を切り替えているのです。速度切り替えジャンパは，電源投入時しか見ませんので，動作中に変えても速度は変わりません。必ず，「マウスジャック2」の電源再投入が必要です。

PS/2マウスは98マウスと違い，CPUを搭載しているインテリジェントデバイスです。PS/2マウスは起動時に初期設定作業が必要なのですが，これは「マウスジャック2」が行っています。そのため，動作中にPS/2マウスの抜き差しはできません。「マウスジャック2」は，PS/2マウスがリセットされたことを検知できないからです。

また，PS/2マウスは個体差・製品ごとの差が非常に激しく，「マウスジャック2」で使用できないものが一部判明しています。これらのサポートは満開ネット（03-3985-6227，6273）などで行っています。

PS/2マウスが使えるようになる変換アダプタだ

MC68060の概要と060turbo.sysの機能

鎌田 誠

満開製作所が製作する新しいマシンの計画では，当面はMC68060を搭載したものを予定しています。そこで，このMC68060というプロセッサがM68000ファミリの中でどのような存在であるのか，その違いと機能を紹介します。また，現行最上位にある，060turboに搭載されているROMでは，どのようにしてX68000と同等の動作をさせているか，について開発者の立場から紹介します。

MC68060の概要

MC68060はモトローラのM68000ファミリのなかで最上位に位置するマイクロプロセッサです。命令キャッシュやデータキャッシュがあるのは当然として，2つの命令を同時に実行するスーパースカラ構造，多段パイプライン処理による分岐命令のオーバーヘッドを大幅に軽減する分岐キャッシュ，FPU，MMUなどを備えています。MC68060は，M68000ファミリの命令セットを継承しながらRISCアーキテクチャに匹敵するパフォーマンスを発揮するMPUなのです。

●スーパースカラ

MC68060の内部では，68000CISCの可変長の命令がIFU（命令フェッチユニット）にある4段からなる命令フェッチパイプラインによって固定長の命令に読み替えられてから，96バイトのFIFOを経由して4段からなるOEP（命令実行パイプライン）に分配され，実行されます。

OEPは，すべての命令を処理できるpOEP(primary-)と，比較的簡単な整数命令を処理できるsOEP(secondary-)の2本があり，いくつかの条件を満たす組み合わせの場合に2つの命令を同時に実行することができます。これをスーパースカラと呼びます。命令の組み合わせの条件は，命令ごとに定められているスーパースカラ分類に従って判定されます。この判定をディスパッチテストと呼びます。

比較的簡単な整数命令の所要時間は1クロックに統一されていますが，スーパースカラの効果で1命令の見掛けの所要時間が0.5クロックになることがあります。たとえば，

```
ADD.L    D0, D2
ADD.L    D1, D3
```

という命令列は10MHzのMC68000で1.6μsかかりますが，50MHzのMC68060では20nsで完了します。これで80倍の差ですが，メモリアクセスを伴う命令ではさらに差が開きます。

余談ですが，「MULU.W Dy, Dx」という整数乗算命令の所要時間は，MC68000で70クロック，MC68030で最大28クロック，MC68040で15クロック，MC68060では実に2クロックです。

●分岐キャッシュ

IFUに過去256回の分岐命令の位置と分岐先のペアを記憶できる分岐キャッシュがあり，これを使って分岐予測が行われます。分岐キャッシュがヒットした分岐命令はOEPに入らないので，実質0クロックで処理されることになります。

分岐予測によってパイプラインの乱れが最小限に食い止められるだけでなく，IFUで処理された分岐命令の前後の命令を同時にOEPに取り込むことができるので，分岐キャッシュはスーパースカラの効果を助長しているといえます。

●FPU（浮動小数点ユニット）

MC68060は，MC68030で使われていたコプロセッサ（MC68882）と同様の浮動小数点演算を行うことができるFPUを内蔵しています。ただし，一部の浮動小数点命令（パック10進数や超越関数など）は実装されておらず，それらはソフトウェアエミュレーションによって処理されます。当然，ソフトウェアエミュレーションであってもMC68882よりも高速です。

●MMU（メモリ管理ユニット）

MC68060はMMUを内蔵しているので，論理アドレス（プログラムから見える見掛けのアドレス）と物理アドレス（メモリやI/Oに割り振られた実体のアドレス）をソフトウェアで組み替えることができます。

論理アドレス空間と物理アドレス空間をそれぞれ4KBまたは8KBずつ区切り，それぞれの1つひとつの領域を論理ページと物理ページと呼びます。各論理ページについて，対応する物理ページの先頭の物理アドレスとその他の属性（物理ページの有無，ライトプロテクト，スーパーバイザプロテクト，キャッシュモードなど）を指定できるようになっています。この指定は木構造のアドレス変換テーブルに記述します。なお，アドレス変換テーブルはユーザーモード用とスーパーバイザモード用に別々のテーブルを指定できるようになっています。

アドレス変換テーブルをメモリ上に構築し，その先頭アドレスをルートポインタと呼ばれるコントロールレジスタに設定したうえでアドレス変換コントロールレジスタを操作することで，アドレス変換が有効になります。

アドレス変換テーブルはメモリ上に存在するので，メモリをアクセスするたびにアドレス変換テーブルを参照していたのではメモリを2回アクセスすることになって非常に効率が悪くなります。そこでMC68060には，命令アクセスとデータアクセスのそれぞれについて64ページ分のアドレス変換を記憶しておくことができるATC（アドレス変換キャッシュ）が備えられています。

●命令キャッシュとデータキャッシュ

メモリのアクセスはMPU内部の動作と比べると非常に遅いので，その回数をなるべく減らすために設けられている仕掛けがキャッシュです。一

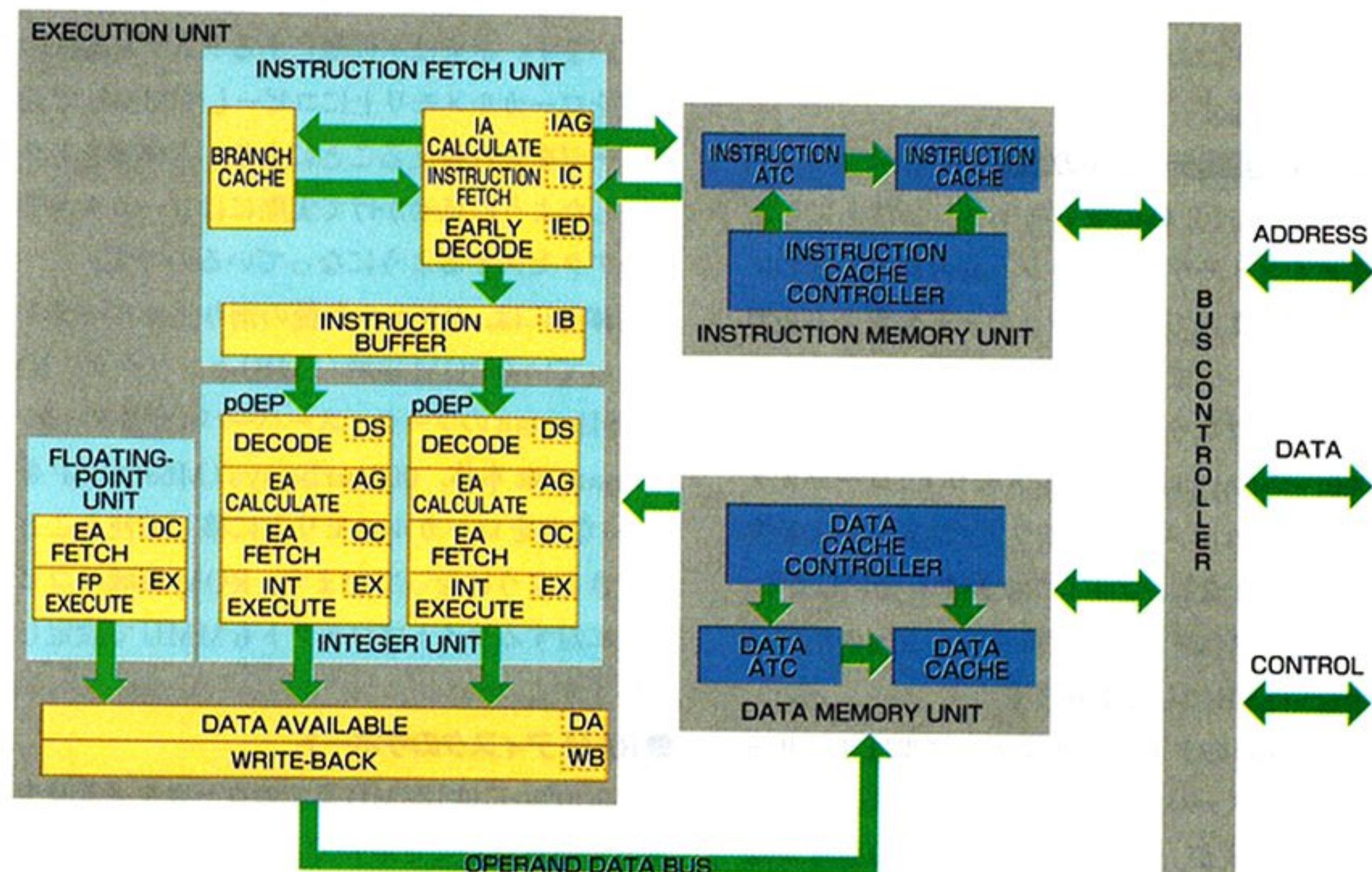



図1 MC68060 Block Diagram

度データを読み出したアドレスについて，そのアドレスとデータのペアをMPU内部のキャッシュに保存しておいて，次に読み出すときは実際にメモリをアクセスせずにキャッシュからデータを取り出すことで済ませてしまうことができます。キャッシュモードによっては，書き込みすらも省略してしまうことがあります。

MC68060は命令フェッチ用のキャッシュとデータアクセス用のキャッシュをそれぞれ8KBずつ持っています。これはMC68040（4KBずつ）の2倍，MC68030（256バイトずつ）の32倍に相当する容量です。

キャッシュの構造はMC68040とほぼ同じで，4-way set associativeという方式が採用されています。命令，データともに16バイトを1ライン，128ラインを1セットとして，それを4セットずつ持っています。

MC68060では，命令キャッシュとデータキャッシュを独立してON/OFFできる以外に，物理ページごとにキャッシュモードを次の(0)～(3)の4つから選択することができるようになっています。

(0)キャッシュ許可・ライトスルー

MC68030と同様の動作をするモードで，メモリへの書き込み動作でメモリとキャッシュの両方が更新されます。DMAで使うデータはこの領域に置きます。

(1)キャッシュ許可・コピーバック

メモリへの書き込みを遅延または省略することで動作を高速化するモードです。このモードではキャッシュ上にダーティデータ（メモリの内容と異なるデータ）が存在し，特にスタックエリアのように同じ場所を何度も更新する領域では大部分のメモリアクセスが省略されるので大きな効果が発揮されます。メインメモリは通常，このモードにします。

ダーティデータをメモリに書き出す動作をキャッシュプッシュと呼びます。キャッシュプッシュはキャッシュが不足したときにMC68060が自動で行う以外に，特権命令でライン単位，ページ単位，キャッシュ全体などの範囲を指定して明示的に行うことができます。

(2)キャッシュ禁止・ストアバッファ禁止

MPUのキャッシュを有効にしてもキャッシュなどが一切効かない領域です。I/O領域は原則としてこのモードにします。

(3)キャッシュ禁止・ストアバッファ許可

ストアバッファは，連続するアドレスへの32ビットに満たない書き込みをまとめて1回で済ませるために書き込みを遅延する機能です。060turboではVRAMをこのモードに設定しています。なお，ストアバッファはライトスルーモードの物理ページに対しても有効です。

060turboのROMについて

060turboには，MC68060で起動できるようにするための専用のROMが含まれています。X68030には新しいROMを装着するためのソケットとジャンパピンが元から用意されているので，060turboではそれを使ってROMを交換するようになっています。060turboのROMも基本的にはX68030のそれと同等のものですが，重要な違いがいくつかあるので説明します。

●MOVEP対策

X68030のROMでは，クロックを調べるところでMC68060がサポートしていないMOVEP命令が使用されているため，そのままではMC68060で起動することができません。このために，060turboを装着する場合にROMの交換が必須になっています。060turboのROMではMOVEP命令を一切使用していません。

●ベクタ対策

X68kのベクタテーブルの未定義ベクタの最上位バイトにはベクタ番号が入っていますが，大容量のローカルメモリを装着していると誤動作する可能性があるため，060turboのROMでは，一部のベクタのベクタ番号を省略するようになっています。

●SCSI入力のパディング

INQUIRYで指定されたバイト数よりも多くのデータを返してしまうSCSI機器が接続されていてその電源が入っていると本体を起動できないという現象を回避するために，INQUIRYなどの入力をパディングするようになっています。

060turbo.sysの機能

060turboには，060turboの能力を最大限まで引き出すための専用ドライバ060turbo.sysが付属しています。ここではこのドライバの機能について説明します。

●アドレス変換テーブルの構築

060turboでI/O領域をキャッシュ禁止にしたままメインメモリをキャッシュ許可にするためには，場所によってキャッシュモードを変える必要があります。つまり，アドレス変換テーブルを構築しなければなりません。

060turbo.sysはローカルメモリ上（ローカルメモリがなければメインメモリ上）にアドレス変換テーブルを作成し，アドレス変換を有効にします。ページサイズはアドレス変換キャッシュの消費量が少ない8KBにしてあります。

MC68060のアドレス変換テーブルはユーザーモード用とスーパーバイザ用に分けて指定することができますが，060turboでは分ける意味がないので，共通のテーブルを使います。

●未実装命令のサポート

MC68060ではM68000ファミリの命令のいくつかが未実装命令になっており，それらはソフトウェアエミュレーションによって実行されます。エミュレーションに必要なプログラムはM68060SP（M68060 SoftwarePackage）としてモトローラが公開しているので，060turboでもそれを利用しました。ただし，MOVEP命令だけ特別扱いになっています。

MC68000に存在する命令でMC68060で唯一サポートされていないのがMOVEP命令です。これもM68060SPで処理できるのですが，060turboではM68000用に作られたプログラムを実行する頻度が高く，しかも一度に大量のMOVEP命令を実行することがあるので，MOVEP命令だけ特別扱いにしてM68060SPよりも3倍以上速いルーチンを作って差し替えてあります。しかし，それでもまだ25MHzのMC68030の約2倍の時間がかかります。MOVEP命令が適したプログラムでは，MPUによって使用するルーチンを分けるといった工夫をするべきです。

なお，後述のアセンブラHAS060.Xには，MOVEP命令をほかの命令の並びに展開する機能があります。

●ローカルメモリの有効利用

従来，X68k用のローカルメモリ（ハイメモリ）は補助的な用途にしか使われていませんでした。ローカルメモリはメインメモリよりもMPUに近い位置に存在するので，メインメモリよりも高速にアクセスすることができます。特に060turboの場合はMPUとX68030本体の動作速度が大きく異なるため，どうしても本体のアクセスにかかる時間がMPUの動作の足枷になってしまいます。

そこで060turboでは，ローカルメモリ（SIMM）を積極的に活用することでX68030本体にアクセスする頻度を減らし，システム全体の高速化を図っています。本来は本体側にあるシステム関係の領域をローカルメモリ上にコピーしてMMUで論理ページを入れ換えることにより，本体側をアクセスしたように見せかけて実際にはローカルメモリをアクセスするようになっているのです。

具体的には，IOCSや各種の割り込みルーチンが入っているROM領域（1MB）と，ベクタ，IOCSとHumanのワーク，スーパーバイザスタック，Human68k本体，060turbo.sys（M68060SPを含む）などをローカルメモリ上に移して使うことができるようになっています。ROM領域のコピー先に対するライトプロテクトもMMUで設定しています。

●RAMディスクのサポート

060turboには128MBまでのローカルメモリを装着することができるので，128MBまで対応したRAMディスクを確保することができます。

RAMディスクの領域をキャッシュ禁止にすることでデータキャッシュを無駄遣いしないようにしてあります。

●Humanのメモリ管理の拡張

ローカルメモリの残りの領域は大容量のハイメモリとしてユーザーに開放しています。

従来のハイメモリと同様の使い方ができるだけでなく，Humanのメモリ管理をハイメモリまで拡張することで普通のプログラムもハイメモリ上で実行できるようになっています。そのためにHumanのメモリ管理部分をまるごと差し替えています。アプリケーションによってハイメモリ上では完全には動作しないことがありますが，SX-WINDOWも大容量のハイメモリ上で動作させることが可能です。

●IOCS _SYS_STATの拡張

X68030ではキャッシュ制御用に使われていたIOCS _SYS_STATを拡張してアドレス変換の設定やスーパースカラのON/OFFなどの制御ができるようになっています。

●DMA対策

060turbo上のローカルメモリ（SIMM）はX68030本体のDMAやMach-2のバスマスタ転送などでアクセスすることができません。そこで，DMA転送が発生すると思われる処理のいくつかにDMA対策を施しています。

DMA転送が発生する可能性があるデバイスドライバにDMA転送できない領域（ハイメモリなど）が指定されたとき，あらかじめ確保してあるDMA転送できる領域を媒介して転送するようになっています。

SCSI関係のIOCSコールでは，DMA転送できない領域が指定された場合は一時的にMPU転送を使用するようになっています。

AD PCM関係のIOCSコールでは，PCM8.Xにならい，2つのバッファを交互に使ってDMAを継続モードで駆動することで，ローカルメモリ上のADPCMデータを再生できるようになっています。

DMAMOVなどのIOCSコールはMPU転送になります。

●XCのライブラリの対策

XCのライブラリでヒープの管理にアドレスの最上位バイトがフラグに使用されているため，アドレスが16MBを超えるハイメモリ上ではXCのライブラリを使用しているプログラムをそのまま実行することができません。

XCのライブラリを使用しているプログラムは市販ソフトなどにも多いことを考慮して，プログラムをロードしたときに自動的にXCのライブラリが使用されていないかどうか調べ，使用されていたらヒープを管理しているルーチンを検索して自動的にパッチを当てるようになっています。

060turboにおけるソフトウェア開発環境

最後に，060turboで動かすプログラムを作る際に留意すべきことをまとめておきます。

●キャッシュ対策

自己展開や自己改変を行うプログラムは，改変後に改変した領域に対して明示的にデータキャッシュのプッシュと命令キャッシュの無効化を行わなければコピーバックモードの領域で動作しません。特にベクタを乗っ取って元のベクタをJMP命令の絶対アドレスオペランドに叩き込んでいるプログラムが正常に動作しなくなることがあるので注意が必要です。

キャッシュのON/OFFやキャッシュの無効化を行う手順は，IOCS _SYS_STATを用いていれば従来どおりで問題ありませんが，キャッシュコントロールレジスタを直接叩いているプログラムは修正する必要があります。

●未定義ベクタの判定

ベクタの最上位バイトにベクタ番号が入っているかどうか調べただけでは未定義かどうかを判別できないことがあります。

●ハイメモリの存在

ハイメモリを使用していると，12MB以上のアドレスにメモリブロックが存在することが考慮されていない常駐プログラムなどが誤動作することがあります。

●分岐予測エラー

分岐予測エラーとは，分岐命令でない場所で分岐キャッシュがヒットした場合に通知されるエラーです。MC68000用に最適化されたプログラムで稀に発生することがあります。このエラーは，キャッシュコントロールレジスタを操作して分岐キャッシュをクリアするだけで継続できます。

●アセンブラ

アセンブラHAS060.XがMC68060の全ての命令に対応しており，ソフトウェアエミュレーションになる命令を通知する機能などを持っています。

●デバッガ

デバッガDB.XやソースコードデバッガSCD.XはそのままではMC68060上で動作しません。これらをMC68060上で動作するようにするパッチが060turboの添付ディスクに入っています。

●NOP命令

MC68060ではNOP命令はパイプラインの掃除をする命令になっているので，ほかの命令と比べて時間がかかります（9クロック）。単にプログラムの途中にできた隙間を埋めるだけのときは，NOP命令ではなくて「MOVEA.L A0,A0」を使いましょう。

●最適化

MC68060の命令の所要クロック数は比較的数えやすいので，コンパイラに頼らなくても手で最適化することができます。

ここでは詳しく説明しませんが，キャッシュのヒット率やスーパースカラのディスパッチテストなどを考慮して，1クロックずつでも削りましょう。VRAMの書き込みウェイトの有効利用など，従来とは違うアプローチからの最適化もできます。

●速すぎても困るもの

FM音源レジスタの書き込みが速くなりすぎて正しく演奏できなくなることがあるなど，必要に応じてウェイトを追加しなければならないプログラムもあります。

おわりに

いつもの癖で，上級者向けのマニュアルのような文章になってしまいました。読みにくくてすみません。

MC68060は，M68000ファミリの命令セットとRISCエンジンが見事に融合した非常に美しいMPUです。スーパースカラや分岐キャッシュの効果は従来のM68000ファミリのプログラムに対しても有効に作用するので，MC68060専用のコンパイラを必要としません。もちろん，MC68060用に最適化すればさらに速くなります。

最近はアセンブラを使える人が少なくなっていますが，MC68060はアセンブラでプログラムを書く意欲を湧きたたせ，プログラムを作る楽しみを与えてくれる，最強のマイクロプロセッサだと思います。

なぜ060なのか？

さて，新しいパソコンを作るというときに，なんでMC68060を使うのかというのが納得できない人もいるのではないかと思う。理由はわからないでもない。

が，MC68000と比べると劇的に速い。PCでいうと286マシンから並クラスのPentiumに乗り換えたくらいの差はあるだろう。なんやかんやいっても速度は十分に速い。

移植，流用できるツールがあり，開発環境がすでにあるという事実は非常に大きな意味を持つ。X68000上でのクロス開発はきわめて容易だろう。現実問題として考えると，その他のCPUでは開発にかかれる人間が極端に限定されてしまう。

ひとたび，基本部分が動き出してしまえば，開発者は格段に増えてくるだろう。

メーカーサイドの人間以外がパソコンを設計して作ろうとする……なんてのは普通に考えるとそうそう実現するわけはない。X68000とMC68060。この組み合わせだからこそ，実現できることというものもあるのだ。

高速なRISC CPUとかもよいだろうが，アセンブラレベルでいじりにくいマシンだと魅力も半減する。そうは思わないか？　第6世代32ビットCPUがフルに回るとどれくらいの性能になるのか見てみたくはないか？（S.N.）

表1 MC68Kシリーズでの命令対応一覧

ニーモニック	68000	68020	68030	68040	68060
ABCD	○	○	○	○	○
ADD	○	○	○	○	○
ADDA	○	○	○	○	○
ADDI	○	○	○	○	○
ADDQ	○	○	○	○	○
ADDX	○	○	○	○	○
AND	○	○	○	○	○
ANDI	○	○	○	○	○
ANDI to CCR	○	○	○	○	○
ANDI to SR[1]	○	○	○	○	○
ASL、ASR	○	○	○	○	○
Bcc	○	○	○	○	○
BCHG	○	○	○	○	○
BCLR	○	○	○	○	○
BFCHG		○	○	○	○
BFCLR		○	○	○	○
BFEXTS		○	○	○	○
BFEXTU		○	○	○	○
BFFFO		○	○	○	○
BFINS		○	○	○	○
BFSET		○	○	○	○
BFTST		○	○	○	○
BGND					
BKPT		○	○	○	○
BRA	○	○	○	○	○
BSET	○	○	○	○	○
BSR	○	○	○	○	○
BTST	○	○	○	○	○
CALLM		○			
CAS、CAS2		○	○	○	○、3
CHK	○	○	○	○	○
CHK2		○	○	○	3
CINV[1]				○	○
CLR	○	○	○	○	○
CMP	○	○	○	○	○
CMPA	○	○	○	○	○
CMPI	○	○	○	○	○
CMPM	○	○	○	○	○
CMP2		○	○	○	3
cpBcc		○	○		
cpDBcc		○	○		
cpGEN		○	○		
cpRESTORE[1]		○	○		
cpSAVE[1]		○	○		
cpScc		○	○		
cpTRAPcc		○	○		
CPUSH[1]				○	○
DBcc	○	○	○	○	○
DIVS	○	○	○	○	○、3
DIVSL		○	○	○	○
DIVU	○	○	○	○	○、3
DIVUL		○	○	○	○
EOR	○	○	○	○	○
EORI	○	○	○	○	○
EORI to CCR	○	○	○	○	○
EORI to SR[1]	○	○	○	○	○
EXG	○	○	○	○	○
EXT	○	○	○	○	○
EXTB		○	○	○	○
FABS				○、2	○、2
FSABS、FDABS				○、2	○、2
FACOS				2、3	2、3
FADD				○、2	○、2
FSADD、FDADD				○、2	○、2
FASIN				2、3	2、3

ニーモニック	68000	68020	68030	68040	68060
FATAN				2、3	2、3
FATANH				2、3	2、3
FBcc				○、2	○、2
FCMP				○、2	○、2
FCOS				2、3	2、3
FCOSH				2、3	2、3
FDBcc				○、2	2、3
FDIV				○、2	○、2
FSDIV、FDDIV				○、2	○、2
FETOX				2、3	2、3
FETOXM1				2、3	2、3
FGETEXP				2、3	2、3
FGETMAN				2、3	2、3
FINT				2、3	○、2
FINTRZ				2、3	○、2
FLOG10				2、3	2、3
FLOG2				2、3	2、3
FLOGN				2、3	2、3
FLOGNP1				2、3	2、3
FMOD				2、3	2、3
FMOVE				○、2	○、2
FSMOVE、FDMOVE				○、2	○、2
FMOVECR				2、3	2、3
FMOVEM				○、2	○、2、3
FMUL				○、2	○、2
FSMUL、FDMUL				○、2	○、2
FNEG				○、2	○、2
FSNEG、FDNEG				○、2	○、2
FNOP				○、2	○、2
FREM				2、3	2、3
FRESTORE[1]				○、2	○、2
FSAVE[1]				○、2	○、2
FSCALE				2、3	2、3
FScc				○、2	2、3
FSGLDIV				2、3	2、3
FSGLMUL				2、3	2、3
FSIN				2、3	2、3
FSINCOS				2、3	2、3
FSINH				2、3	2、3
FSQRT				○、2	○、2
FSSQRT、FDSQRT				○、2	○、2
FSUB				○、2	○、2
FSSUB、FDSUB				○、2	○、2
FTAN				2、3	2、3
FTANH				2、3	2、3
FTENTOX				2、3	2、3
FTRAPcc				○、2	2、3
FTST				○、2	○、2
FTWOTOX				2、3	2、3
ILLEGAL	○	○	○	○	○
JMP	○	○	○	○	○
JSR	○	○	○	○	○
LEA	○	○	○	○	○
LINK	○	○	○	○	○
LPSTOP					
LSL、LSR	○	○	○	○	○
MOVE	○	○	○	○	○
MOVEA	○	○	○	○	○
MOVE from CCR		○	○	○	○
MOVE to CCR	○	○	○	○	○

ニーモニック	68000	68020	68030	68040	68060
MOVE from SR[1]	4	○	○	○	○
MOVE to SR[1]	○	○	○	○	○
MOVE USP[1]	○	○	○	○	○
MOVE16				○	○
MOVEC[1]		○	○	○	○
MOVEM	○	○	○	○	○
MOVEP	○	○	○	○	
MOVEQ	○	○	○	○	○
MOVES[1]		○	○	○	○
MULS	○	○	○	○	○、3
MULU	○	○	○	○	○、3
NBCD	○	○	○	○	○
NEG	○	○	○	○	○
NEGX	○	○	○	○	○
NOP	○	○	○	○	○
NOT	○	○	○	○	○
OR	○	○	○	○	○
ORI	○	○	○	○	○
ORI to CCR	○	○	○	○	○
ORI to SR[1]	○	○	○	○	○
PACK		○	○	○	○
PBcc[1]					
PDBcc[1]					
PEA	○	○	○	○	○
PFLUSH[1]			○、5	○	○
PFLUSHA[1]			○、5		
PFLUSHR[1]					
PFLUSHS[1]					
PLPA					○
PLOAD[1]			○、5		
PMOVE[1]			○		
PRESTORE[1]					
PSAVE[1]					
PScc[1]					
PTEST[1]			○	○	
PTRAPcc[1]					
PVALID					
RESET[1]	○	○	○	○	○
ROL、ROR	○	○	○	○	○
ROXL、ROXR	○	○	○	○	○
RTD		○	○	○	○
RTE[1]	○	○	○	○	○
RTM		○			
RTR	○	○	○	○	○
RTS	○	○	○	○	○
SBCD	○	○	○	○	○
Scc	○	○	○	○	○
STOP[1]	○	○	○	○	○
SUB	○	○	○	○	○
SUBA	○	○	○	○	○
SUBI	○	○	○	○	○
SUBQ	○	○	○	○	○
SUBX	○	○	○	○	○
SWAP	○	○	○	○	○
TAS	○	○	○	○	○
TBLS、TBLSN					
TBLU、TBLUN					
TRAP	○	○	○	○	○
TRAPcc		○	○	○	○
TRAPV	○	○	○	○	○
TST	○	○	○	○	○
UNLK	○	○	○	○	○
UNPK		○	○	○	○

注：
1. 特権命令
2. MC68EC040、MC68LC040、MC68EC060、MC68LC060には適用されない
3. MC68040とMC68060ではソフトウェアでサポートされる
4. この命令はMC68000とMC68008では特権命令ではない
5. MC68EC030には適用されない
6. MC68060とMC68040ではインプリメントされていないデータタイプでは浮動小数点演算にソフトウエアサポートが必要。MC68060ではインプリメントされてない実効アドレスに対してもソフトウェアの補助が必要

WWWブラウザ
WebXpression

満開製作所：山口光樹

電脳倶楽部別冊16号以来X680x0でWWWを楽しむ人が増えてきました(別冊16号はインターネット特集号でした)。しかし現行のX680x0用ブラウザはキーボード操作でなおかつ画像が表示されないため世間一般でのWWWのイメージからはほど遠いようです。そこで一発！　ど根性，画像表示可能なブラウザを作成してみました。もちろん10MHz機でも楽々起動，マウスひとつで楽々操作可能です。

インストール

パスの通ったディレクトリに実行ファイルWebXpression.xとコンフィグファイルWebXpression.cnf，アドレス帳ファイルAddressBook.htmをコピーします。次に環境変数WEBCACHEにダウンロードしたファイルを保存するディレクトリ名を設定します。

set WEBCACHE=A:¥WEBCACHE¥

JPEGファイルの展開にJPEGED.R(電脳倶楽部別冊16号掲載)を使用するのでパスの通ったディレクトリに置いてください。また，MFD.X(電脳倶楽部51号掲載)やhfont.r(激光電脳倶楽部1号掲載/Ko-Windowアーカイブ内)などで12ドットフォント「要町」を組み込むと読みやすく，かつ高速に動作するようになります。以上で完了です。

インターネット上のファイルを表示させるためにはTCP/IPドライバのインストールが必要ですが，誌面の都合からここでは割愛します。詳細は電脳倶楽部別冊16号などを見てください。ディスク上のファイルを読むだけなら，ドライバがなくても動作しますので試してみましょう。

操作法

コマンドラインなどから，

WebXpression [option] [URL]

として起動します。URL未指定時にはAddressBook.htmを表示します。

例)

WebXpression http://www.mankai.co.jp/

URLはhttp://から指定してください。また，ローカルファイルを閲覧する場合はfile://ファイル名としてください。URLの先頭がhttp://でもfile://でもない場合，file://として扱います。

操作法は基本的にDSHELLを拡張した操作体系になっています。

左ボタンをクリック：正方向スクロール
そのままマウスを下にドラッグ：正方向高速スクロール
右ボタンをクリック：逆方向スクロール
そのままマウスを上にドラッグ：逆方向高速スクロール
両ボタンクリック：ひとつ前のページへ戻る
下線の引いてある文字列(リンク)の上でマウス左ボタンをクリック：リンク先のテキストを表示　一度表示したリンクは黒で，まだ表示していないリンクは赤で表示されます。
[BREAK]：終了する

なお，現在制作中につき画面右側に表示されているボタンは機能しません。今後の展開にご期待ください。バージョンアップ版は電脳倶楽部に随時掲載していく予定です。

注意点

68000機ではネットワークからファイルを受信中に画面を書き換えるとデータを取りこぼすことがあるため，WebXpression.cnfのhold-onlineを1(受信中は画面を書き換えない)に設定してください。こうすると受信中はマウスカーソルが消え，[ESC](受信の中断)以外の入力を受け付けなくなりますが，データを取りこぼすことがなくなります。

技術解説

WWWブラウザは私たちの言葉でいえば通信機能つきのテキストシェルです。通信部分のGetFile.cはHTTP(Hyper Text Transfer Protocol)によりファイルを取得し，キャッシュに保存します。ファイルを取得するにあたってはまずファイルの情報(ファイルサイズやタイムスタンプ)のみを取得し，更新されていた場合に限ってネットワーク上からファイル本体を取得するようにしています。

テキストビュア部分はその他全部です。コード的な比率からいえば全体の9割近くがテキストビュアとなっています。取得したHTMLファイルは行ごとに折り返して表示します。HTML中に画像が貼り込まれていた場合はFIFO形式のバッファに画像のURLが記録され，その後，逐次ネットワークから取得します。画像の画面上での大きさは取得前にはわからないため(わかる場合もある)，そのたびにHTMLの折り返しを行います。一般的なHTMLファイルではこれに要する時間は微々たるものなのでユーザーにストレスを感じさせることはないでしょう。

また，マウス/キー入力および画面描画はすべて割り込みで動作します。そのため上述のHTML折り返し中もJPEG展開中も常に入力を受け付け，画面の更新を行えます(スムーズスクロール中にバックグラウンドでJPEGED.Rが動いています)。ですが，低クロック機で通信中にこれを行うとやはり不具合があるようなのでこれを禁止することも可能です(前述のhold-online)。作者の環境(060turbo)ではいったん通信が途切れたあと，きちんと再開してくれます。

ベタのHTMLは，タグを解析したり，余分な複数のスペースをひとつにまとめる必要があったりと，リアルタイムに処理するにはいささか重い形式です。X680x0では走査線が40本走る前に描画を完了させないと処理落ち(見た目ではスクロールのガタつき)となって現れてしまうため，HTMLファイルは折り返すとともに高速表示可能な内部形式へと変換されます。この内部形式もまた可読なテキストファイルとなっています。しかもデバッグモードではこの内部形式をダンプすることが可能なので，バグが発生しても原因を究明することが非常に楽になっています。完全に速度を優先するのであれば内部形式はバイナリにするべきなのですが，デバッグの容易さを優先して現在の形式を採用しています。動かない高速なプログラムよりも多少遅くとも動くプログラム，ですよね。それでもテキストのみのHTMLではhfont.rなどを使用していれば10MHz機でもほとんど処理落ちしない程度の速度は叩き出しています。

最後に

付録CD-ROMには実行ファイルとともに詳細なドキュメントが収録されてますのでそちらもあわせて見てください。当然(？)のことながらソースリストも収録されています。視点/マウス移動の少ないDSHELLスクロール，省資源設計，そしてソースリスト添付というX680x0の醍醐味を感じさせる1本ではないかと作者自身は思っているのですが，これを見ているあなたはどう感じたでしょうか？

それではまたの機会に。

パソコン通信のすすめ

猪狩友則

始まりは，休刊からだった

周囲に，X680x0のユーザーはいただろうか。そして，いまでも使っている人が周りにいるだろうか。私の周りでは当時，NECのPC-98を使っている奴が多かった。金のある奴はMacintoshを使っていた。私にとって，当時情報といえば，Oh!Xと電脳倶楽部だけだった。全盛だった当時なら，周りにユーザーがいてやり取りができた方もいただろう。しかし，最近になってから以前使っていましたという方に会うことはあっても，当時ユーザーの知り合いは皆無だった。やがて，電脳倶楽部の購読をやめてしまい，Oh!Xも休刊になった頃，X68000に触れることもだんだん少なくなっていた。そんなとき，郊外のパソコンショップで，投げ売りになっていたモデムを買った。

通信のことはさっぱりわからず，当然相談できる人などは周りには皆無で，本屋で適当に通信関係の本を買ってくる。いま考えると，無駄に買った本のベスト3に入るくらい内容が薄い本だったが，NIFTY SERVEのイントロパックがついていという意味では，影響のある本だったともいえる。そのイントロパックを見ながら，接続させたときのピーガーというネゴ音を聞いたとき，妙な感動があったのは，いまでも忘れられない。

いつの間にか，毎日のようにアクセスするようになり，気がつくと，フォーラムでスタッフをやってたりする。これで，Oh!Xが休刊になっていなかったらどうなっていたのやら。

図1　NIFTY SERVEのFSHARPへアクセス

情報こそがすべて

現在でも，X680x0を使っている人が身近にいるのなら，いいだろう。いまどきの容易にインターネットに接続できるパソコンを持っている人は，それを使って情報収集してもよいだろう。さて，そのどちらにも当てはまらない人はどうすればよいのだろうか。最近ではX680x0でインターネットに接続できるらしいと，知ってはいても，実際の手順はわからない。周囲には，X680x0はおろか，パソコンを使いこなしている人間すら皆無。そんな人，結構多いのではないだろうか。

昔は，皆が皆そうだったから，それでも気にならなかっただろうが，これだけ他機種の情報が氾濫していると孤軍奮闘は辛く，使い続けることに疑問は持たないにしても，ほかのマシンに浮気したくなるだろう。そこまで大げさでなくても，人間は結構弱いもので，自分ひとりだけというのを，なかなか続けられないものである。少なくとも私はそうだ。何度，捨ててしまおうと思ったことだろう。

つまり，ほかに誰か使っている人がいるのを感じていられるというのは，使い続ける上で大きなウェイトを占めると思う。Oh!Xはドラゴンだったが，ユーザーが皆ドラゴンになれたわけではないのだ。もちろん，実際に，ドラゴンを作り出していた雑誌だったと思う。仕事であれば，古くても嫌いでもタコでも使うが，趣味でやっていると，周囲の雑音が余計に気になるものだと思う。

だからといっていまから周囲に，X680x0を買わせたところでどうなるものでもない。普通にプロバイダに加入して，X680x0でインターネットの世界に飛び出すのもよいだろう。しかし，おすすめするのはインターネット全盛の時代，いまさらと思われる方も多いだろうが，パソコン通信である。

商用ネットでもいいし，近くに草の根BBSがあれば，そちらを利用してもよいだろう。手前味噌ではあるが，NIFTY SERVEをおすすめする。FSHARP(SHARP Products Users' Forum)を利用するのだ。商用ネットは接続料金がかかるからと遠慮していた方も多いだろうが，以前と比べれば驚くほど安くなっている。人にもよるだろうが，普通に使うのであれば，月にすれば，タバコ代程度だ。

FSHARPを利用しなくても，メールアドレスを持てるというだけでメリットは大きい。なんだかんだいっても，インターネットでいちばん使うサービスはメールだからだ。さて，パソコン通信で必要になるのは，本体のほかにモデムとターミナルソフトで，モデムも最近は世の中のインターネットの普及により値段が下がっている。

パソコン通信は，インターネットを利用するときと違って接続が容易というのが，魅力のひとつである。つまり，トラブルが少ないということである。また，商用ネットはたいていプロバイダも兼ねているので，インターネットを使いたくなったら，使うことができる。また，NIFTY SERVEの場合，FSHARP以外にも，X680x0関連の話題を扱っているところがあるようなので，そちらを回ってみてもよいだろう。

フォーラム

ファイルシステムのような階層構造を思い浮かべてほしい。ファイルとディレクトリのイメージだ。パソコン通信のサービスの構成に限らず理解しやすいようにすると，自然にこうなる。いろいろな機器の操作法はこのようになっていると思う。

NIFTY SERVEはテーマごとに分かれたフォ

表1　FSHARP内の会議室の様子

番号	発言	(未読)	最新	会議室名
1	63	(0)	07/03	【INFO】★　FSHARP INFORMATION　★
2	1138	(0)	07/03	【HELP】----ビギナー質問/自己紹介----
3	158	(0)	05/25	【FAQ!】☆ よくあるQ＆A集 編集室 ☆
4	4466	(11)	07/04	=-=-=-=-=-=- 壁 WALL WALL -=-=-=-
5	2809	(10)	07/04	【HARD】 ■ メビウスユーザーエリア ■
6	1571	(0)	07/03	【HARD】 ■ メビウス　改造 の 部屋 ■
7	349	(0)	06/30	【SOFT】 □ソフト全般（含 Windows） □
8	290	(0)	06/27	【HARD】 □ 通信/LAN/IrDA/周辺機器 □
9	679	(0)	07/03	【HARD】 □ 携帯端末　ザウルス etc □
10	572	(6)	07/04	・・・・ FSHARP オークション ・・・・
11	4380	(2)	07/04	【TALK】 《　FREE 雑談/交流　》
12	175	(0)	06/10	【TALK】 《 シャープ ユーザーの声　》
13	159	(0)	07/03	【HARD】 ◆　ＡＶと家電製品エリア　◆
14	90	(0)	06/30	【NEW!】 ◆デジカメ《VELC2・MDPS1》◆
15	963	(5)	07/04	【NEW!】 『 新機種 メビウス　PC-PJ1 』
16	901	(2)	07/04	【SPL!】 《 特設 R.C.C olosseum 》
17	138	(0)	06/09	【HARD】 《　ＭＺ　ユーザーエリア　》
18	223	(0)	07/03	【HARD】 《　Ｘ１　ユーザーエリア　》
19	1672	(0)	07/03	【HARD】 《　X68　ユーザーエリア　》

ーラムの階層構造になっており，FSHARPはそんなフォーラムのなかのひとつだ。FSHARPでは，シャープ関連の話題を扱う場所としてX680x0やX1にMZに限らず，シャープ製品一般を広く扱っている。

さて，それぞれのフォーラムには，会議室というものがある。会員全員で交換日記をやっているのをイメージしてもらうとわかりやすいと思う。交換日記と違うのは，書くのも読むのも自分の好きな時間にできるということだ。

質問があれば，そこに書き込んでおけば，誰か答えてくれるかもしれない。自分がわかる質問があれば，答えてあげるとよいだろう。なにか発見したら，発表してもいい。そう，コミュニケーションの場なのだ。

やはり，パソコン通信を継続するのに，それは大きなウェイトを占めるのである。

その会議室は，話題によって分かれていて，X680x0の話なら，19番だ。最近の話題としては，やはり，インターネットの接続の話や，Windowsマシンとのデータファイルのやり取りについてが多い。もちろん，フォーラムそして，会議室のテーマにあった内容であれば，各自の好きに書いて構わない。

ゲームソフトROBOT CONSTRUCTION(Electric Sheep)に関しての特設会議室が，16番にあり，定期的に大会が行われている。腕に覚えがあるRCerは参加してみてはいかがだろう。ちなみに，このソフトはまだ手に入るようである。ほかに，X1，MZ系，ノートパソコンのMebiuや，ザウルスやポケコンを扱う会議室もそれぞれ用意されている。自分の興味があるところを覗いてみるとよいだろう。シャープ製であれば，掃除器だろうがラジカセだろうが話をできる家電の会議室もある。X680x0ファンというより，SHARPファンという方も少なくないハズだ。

発言せず，読むだけの人は，ROM(Read Only Member)と呼ばれる。パソコン通信関連の本や雑誌の記事によっては，その存在を嫌い，非難めいたことが書かれていることもあるが，気にすることはない。読むだけでもいいのだ。場所によっては嫌われるところもあるのだろうが，少なくともFSHARPでは，まったく問題ない。人によってパソコン通信に割ける時間は違うだろうし，書くのが苦手な人もいるだろう。自分の好きなときに，好きなように参加できるのがパソコン通信のよいところなのだ。頻繁に書き込む人も，ROMの人も上下はないのだ。

古い発言はライブラリに会議室ごと，発言番号に分けられて，登録されている。バックナンバーが手軽に読むことができるというわけだ。数年にわたるそれは，ちょっとやそっとの時間では読み尽くせないだろうが，そこから自分がほしい情報を探し出したときの喜びはひとしおである。私も，何度となく困ったときにここから探し出し，助けられている。

このライブラリというのは，本来，プログラムやデータを，保管しておくところだ。気に入ったソフトウェアがあれば，ダウンロードして使ってみるとよいだろう。自分で作ったものをアップロードして，ほかの人に使ってもらえるようにするのもよいだろう。現在登録されている数は，X680x0用が全部で2,078件，X1用が33件，MZ用は209件となっている。ほかにも，メビウス用のとして，シャープより提供いただいたものもある。

ライブラリは会議室同様アップロードされている種類によって，複数に分かれている。なお，FSHARPに登録されているライブラリの一覧はCD-ROMに収録されているので参照してみてほしい。NIFTY SERVEのFSHARP全体の構成もわかるようになっているので，どんな会議室があるかなども確認してみてほしい。

先述のROMと似た存在として，DOM(Download Only Member)と呼ばれる人たちがいる。要するに，会議室で書くわけでもなく，ひたすらダウンロードを行う会員のことである。確かに回線数が少ない草の根BBSだと，回線が占有されるので嫌われる存在だろう。しかし，NIFTY SERVEは商用ネットであるため，そういった問題はない。コレクションのようにダウンロードをするのも，楽しみ方のひとつといえる。ソフトを目的に，通信をやってもよいのだ。気に入ったら，作者の方にメールを出したり，会議室で感想を書くとよいだろう。が，それさえも強制されるようなものではない。

リアルタイム会議室

リアルタイム会議室というものがある。いわゆるチャットという奴だ。普通の会議室の書き込みには，おのおの時間差が存在する。それに対してリアルタイム会議室は，同時にその場にいて，文字を使ってお喋りをするのである。雑談も多いのだが，非常にためになるときもある。以前，ちょうどX680x0ユーザーだけになったときに，MOを使ったファイルのやり取りのトラブルの話になった。お互いに知っていることを書き，お互いに質問をぶつける。

その場でのやり取りなので，非常に話が早く，会話でありながら，文字で認識されるため非常に勉強になる。このようなことは，チャットならではではないだろうか。やり方がわからないまま入ってしまった場合でもその場で手取り足取りで教えてくれるだろう。自分が所有している，もう1台のパソコンで相手が使っている通信ソフトを立ちあげ，ひとつずつ説明している親切な方もいた。

普通の会議室でもいえることなのだが，会ったこともない人と文字だけでコミュニケーションをする。そう考えると少々不気味に見えるだろうが，1回やってみるとまったく気にならなくなると思う。お互いに知らない，つまり俗世間のしがらみを意識しないで済むということを感じるだろう。現実逃避という見方もあるだろうが，逆に，そういったしがらみを度外視して，対等につきあいができるのもまた，通信のよいところである。が，通信だからといってそれに甘えて，なにをやっても許されるわけではなく，一般社会と同様の部分も多い。この辺は，そう身構える必要はない「習うより慣れろ」だ。

宣伝，宣伝

FSHARPでは，オークション会議室を設けている。出品されたものに対して，ほしい人は金額を書いて入札していくのだ。少しずつ，釣り上げていってもよいし，一気に引き離してもいい。そういった駆け引きも楽しんでもらいたい。使わなくなったけど捨てるくらいなら誰かに使ってもらいたい，そんなものがあれば出品してみるとよいだろう。思いもよらぬ掘り出しものがあるかもしれないのでチェックは欠かせない。

なお，出品できるのは，電気製品に限られている。また，オークションに入札する人のために，出品したものの画像をアップロードできるように専用のライブラリも用意してある。金銭が絡むだけに，トラブルを避ける意味でもルールをきちんと理解してから参加してほしい。

さて，現在FSHARPは，シャープとの間でシャープ製X680x0用ソフトの再配布に関して交渉中だ。許可が下りたものから，順次FSHARPのライブラリで公開していく予定なので楽しみにしていてほしい。ゲームや一般企業のOEM製品を除いたほぼすべてのソフトが半フリー化されて配布可能になる予定だ。

X68000が押し入れで埃をかぶっている方は，ぜひこれを機に，再度使ってみてはいかがだろうか。他の媒体でも紹介できる機会があれば，積極的に紹介していく予定である。できれば今回付属のCD-ROMにSX-WINDOW Ver3.1だけでも入れたかったのだが，調整が間にあわず，それはかなわなかった。とても残念である。

実際にNIFTY SERVEに加入するには，イントロパックを入手して，オンラインでサインアップするとよい。利用方法に関しては，専門の書籍などを参考にしてほしい。

私のハンドル(ネットワーク上の自分で決めるあだ名)は「いがり」である。FSHARPで見かけたら声をかけていただけると嬉しい。

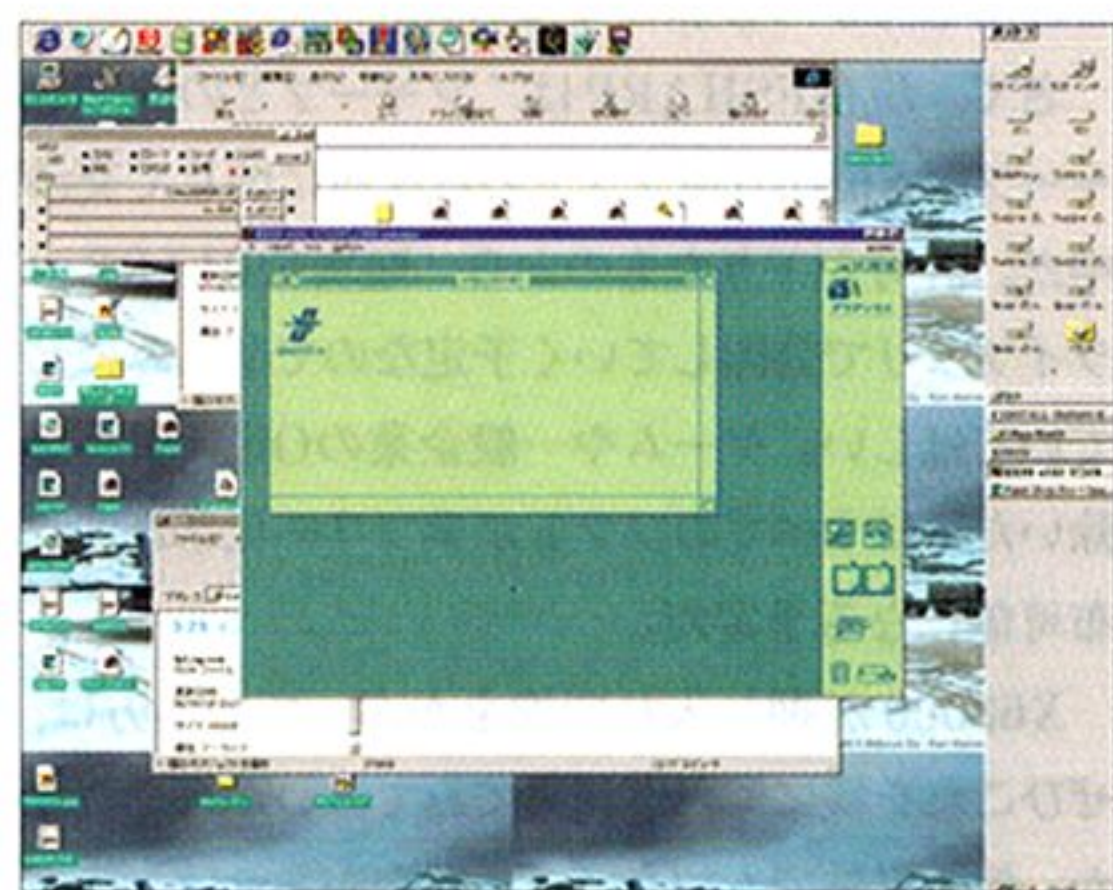
Windows用X68000エミュレータEX68

山口 正

現在ではパソコンを選ぶとなると，ほぼPCかMacのどちらかになりますが，かつてはX68000という魅力的なパソコンがありました。すでに生産は終了していますが，いまでも活発なユーザーの下で独自のハード，ソフトの拡張が行われています。

EX68はこのX68000をWindows95/NTの上でエミュレートするソフトウェアで，その目的は実機と同じ機能をより手軽で便利に動作させることにあります。しかしエミュレータEX68を作る動機は実用に利用しようというよりも，プログラムそれ自体を楽しむことにあります。エミュレータを作る過程では，いまでは簡単にはできなくなった回路図を基に部品を組みパソコンを作り上げていくハードウェア工作の面白さを擬似的に体験できるようにと思われます。

こうして現バージョンの機能は完全ではありませんがインターネット上で公開デバッグを始めて1年ほどになります。その間に寄せられた多くの動作報告やバグレポート，実機の動作解析，機能改善のアドバイスにより，当初想定していた以上の機能を組み込むようになりました。

EX68はソフトウェアによるパソコンのエミュレータですので機能上はパソコンを丸ごと作ることになりますが，パソコンをブラックボックスと見るならX68000上のプログラムからの入出力とWindowsとの入出力の仕様を満たすことでハードウェア上の回路を組み込むことなく，実機と同じに振る舞う仮想のパソコンを実現できます。

画面1　懐かしきビジュアルシェルが！

画面2　もちろんグラディウスもこのとおり

画面3　複数起動も可能だ。重いけど…

ここでの入出力はサブルーチンの呼び出しによるパラメータの受け渡しで実現されます。プロセッサルーチンからのアドレスとデータはリードライトルーチンに渡り，各周辺LSIルーチンへと呼び出しが起こり，最後にはWindowsの入出力デバイスへと渡されます。この結果としてWindowsの中にX68000が再現されます。

X68000ではハードウェアの仕様書から回路図まで公開されていますので，主にこれらの資料を基にエミュレータを設計しますが，ここではEX68のエミュレーションを機能上から3つの部分に分けます。CPUエミュレーションとカスタムチップエミュレーション，周辺機器エミュレーションです。

CPUエミュレーション

最初はX68000のメインプロセッサであるM68000のエミュレーションです。実行されるプログラムからの振る舞いがM68000と同じであれば機能上は十分ですので，プログラミング上のモデルを基にレジスタ，命令デコーダ，外部アクセス，演算，制御などに相当するサブルーチンを作ります。

EX68ではM68000プロセッサの仕様を基にリソースをリスト1のように宣言(抜粋)します。

プロセッサの最初の動作はリセットですので初期化ルーチンdo_reset()を起動し，構造体MPUの初期設定を行います。

リスト1

```
struct{
    ULONG   _d[8];          //データレジスタ D0からD7まで確保
    ULONG   _a[8];          //アドレスレジスタ A0からA7まで確保
    ULONG   _pc;            //プログラムカウンタ
    ULONG   _usp,_ssp;      //スタックポインタ
    ULONG   _sr;            //ステータスレジスタ
    ULONG   _inst_ptr;      //インストラクションポインター
    ULONG   _accessaddr;    //外部アクセスポインタ
    ULONG   func_code;      //ファンクションコード
    struct {
            ULONG           flag;
            ULONG           level;
            ULONG           vector;
            ULONG           reqbit;
    }inter;                 //外部割り込み要求
    struct {
            ULONG       cur;    //プロセッサクロックカウンタ
            ULONG       next_int_cycle;  //多重タイマ割り込みカウンタ
    }cycle;
}MPU;
```

リスト2

```
void do_reset()
{
    MPU._sr = 0x2700;                     //ステータス設定
    MPU._pc = fetl_(0xff0004);            //ROM内の初期PCを設定
    MPU._sp = fetl_(0xff0000);            //ROM内の初期SPを設定
    MPU._usp = MPU._ssp = MPU._sp;        //2つのスタック設定
    MPU.cycle.cur = 0;                    //クロックは0から開始
    MPU.inter.flag = 0;                   //割り込みは無し
    .
    .
    //以下X68000ハードウェアの初期化を行う
}
```

このなかのfetl_(0xff0004)はX68000内のメモリ(アドレスff0004はROM)の読み出しルーチンで，プロセッサエミュレーション部と周辺LSIエミュレーション部を繋ぐインタフェイスになります。同様に書き込みルーチンはwril_(dat, addr)となり，画像メモリのアドレスや，FM音源のLSIのアドレスを渡し，データ書き込みのエミュレーションを行います。

こうしてプロセッサの初期化が終わり，周辺部との読み書きができるようになりましたので，プロセッサは動作を開始します。

ここでは1命令ごとの解釈，実行ループを開始します。

リスト3

```
void instruction()
{
WORD inst;
ULONG clock;
   do {
    if (MPU.inter.flag)                 //割り込みを調べ
            do_interrupt();             //割り込みを受け付ける
    inst=fetw_(MPU._pc);                //1命令読み出し
    clock=decode(inst);                 //1命令解釈，実行
    MPU.cycle.cur+=clock;               //クロックを進める
    chk_timer();                        //タイマーをチェックする
   }while(!X6.io_req);                  //次の命令へ繰り返し
}
```

実際はアセンブラで書き直したうえでデバッグ用のコードや処理速度を上げるコードを組み込んでいます。現状ではエミュレーションの速度が優先されますのでdecode()以下の命令解釈実行ルーチンのほとんどはアセンブラで書かれていて，

Pentium/166MHzならばM68000/10MHz 以上で動作します。

プロセッサの動作は仕様がはっきりとしていますので，デバッグが進むにしたがい，実機との互換性は動作タイミング以外の問題はなくなります。

また，プロセッサエミュレーションにはデバッグ用の機能としてハードウェアエミュレータ(ICE)などでも使われる簡単なアクセスブレークや，アクセスヒストリを組み込み，エミュレータのデバッグに利用します。

一連の動作の記録を取るアクセスヒストリはデバッグ以外にもX68000プログラムの動作解析には特に有用で，プログラム別に同じ機能を実現するさまざまな手法を知ることができます。実際にゲームでは一定の速度で動作させるための方法が何種類も存在します。

これらを知ることにより，より効率的な実装方法を検討し，より軽い動作を実現できます。

周辺LSIエミュレーション

２つ目はプロセッサから操作されるメモリや周辺LSIのエミュレーションです。ここはX68000ハードウェアに対するエミュレーションの中心部分といえますが，M68000プロセッサからのアクセスに対して実機と同じように正しく応答することが重要で，どこまで機能を正確に実装できるかが実機との互換性能を決めることになります。

ここでもX68000の公開資料を基にリソースを宣言(抜粋)し，実機で存在するメモリ，レジスタは同じように確保しています。

リスト4

```
struct {
    UBYTE   *mmem;          //メインメモリを確保
    UBYTE   tmem[512*1024+8];       //テキストメモリ
    UBYTE   gmem[1024*1024+8];      //グラフィックメモリ
    struct {
        union {
            UBYTE       spmem[64*1024+8];    //スプライト
            USHORT      _spmem[(64*1024+8)/2];
            };
        union {
            UBYTE       dspmem[16*1024+8];   //ダブラ用バッファ
            USHORT      _dspmem[(16*1024+8)/2];
            };
    }spr;
    struct {
            int tup,tdown,tleft,tright,sel;  //JOYスティック
    }joy[2];
    struct {
            UBYTE       dat[0x10*0x10];      //MIDIレジスタ
            ULONG       gadr,use,out,enable;
    }midi;
    USHORT  palette[256+256];       //パレットレジスタ
    USHORT  crtc_reg[0x38/2];       //CRTコントローラのレジスタ
    struct {
            USHORT      reg[0x100/2];        //DMACレジスタ
            USHORT      vect;
            int         access,error;
    }dma;
    USHORT  R_joybuff[2];           //ジョイスティックからの入力値
    UBYTE   sram[SRAMSIXE+8];       //SRAM
    UBYTE   R_mfp[0x34+4];          //MFP
    UBYTE   R_fd[8];                //FD コントローラ
    UBYTE   R_adpcm[8];             //ADPCMコントローラ
    UBYTE   R_joy[8+4+4];           //ジョイスティックポート
    UBYTE   rtc_reg[0x20];          //RTC
    UBYTE   sys_reg[16];            //SYS
    UBYTE   vid_reg[6],pvid_reg[6]; //VIDEOレジスタ
    UBYTE   R_area;                          //AREAコントローラ
    UBYTE   rtc_bank;               //RTCのバンクレジスタ
    struct {
            UBYTE       reg[0x100]; //FM音源レジスタ
            short       ym[2][0x100];        //SB FM音源レジスタ
            ULONG       regadr;
            int         enable,sample,pcm8,port;
    }fm;
    ULONG   contrast;               //コントラストレジスタ
    ULONG   sccA_reg,sccB_reg;      //SCC
    ULONG   sccA_seq,sccB_seq;
    UBYTE   sccA_sel,sccB_sel;
}X68;
```

メモリなどの動作は単純ですので実装は簡単ですが，カスタムLSIや汎用のLSIなどは複雑で多くの機能を持っていますので，いまでもすべての実装はできていません。この未実装の中には実機でも機能しない部分や，内蔵時計への書き込みなど省略してよい部分，ハードディスクへの直接の書き込みなどそのまま実装しては危険なもの，実時間での処理が間にあわない機能なども含まれます。

現在はこれら以外のよく利用されている機能から優先して実装していますが，高速なカウンタのように，処理は重いが不可欠なものや，キーボードの通信速度の変更など実際に使われているか確認できていない機能は，軽く単純な動作に換えた形で実装しています(表１)。

これらのLSIでは，基準クロックを基に多くのタイマが動作していますが，EX68ではプロセッサエミュレーションから作成した仮想のクロックを基にした仮想の時間とマルチメディアタイマにより計測した実際の時間から仮想の垂直同期信号を作成し，この２つの時間から内部の基準タイマを動作させ，この基準タイマによってすべてのLSIのタイマを駆動しています。

そのほかX68000では拡張機器となるMIDIボード，メモリボード，ハードディスクなどもエミュレータではソフトで実現することになります。これらは設定により自由に増設でき，実際に手元で使用しているX68000以上の環境が得られています。

周辺デバイスエミュレーション

３つ目はWindowsに接続されたデバイスに置き換えることのできる入出力のエミュレーションになります。入力のなかでマウスとキーボードはユーザーからの入力をWindowsメッセージとして受け取ったあと，自由度はないが互換性の高い周辺LSIへの入力形式に変換する方法と，互換性は低いが使い勝手のよいX68000の基本ソフトウェアであるIOCSのルーチンへ変換する方法のうちから選択できるようにしています。

なかでもキーボードはユーザーの環境によってさまざまなレイアウトのものが使われていますのでキーコードの変換表を設定ファイルにより定義しています。以下のように変換方法に応じてテーブルを定義しています。

ゲームで使われるジョイスティックはWindows用標準のものキーボードに割り当てたもの，NECのPC-98で使われているデジタル入力のものから

表１　現在の実装状況

＜実装終了＞	
M68000プロセッサ	(タイミング以外は終了)
メインメモリと領域保護	(最大12MB)
ＦＭ音源	
ADPCMの再生	(一括変換)
DMA ch2，ch3	(メモリ間転送とADPCM転送)
ハードディスク	(iocs互換)
フロッピーディスク	(iocs互換)
MFP	(割り込みコントローラ)
MFP	(タイマーABCD)
MFP	(USARTキーボード)
MFP	(GPIPパラレル入力)
SRAM	(EX68終了時にファイルsram.ramへセーブする)
RTC	(時計の読み出し)
SCC	(マウスポートの入力)
CRTC	(レジスタすべて)
スプライトダブラ	(16倍表示)
ラスタースクロール	(テキスト，グラフィック，スプライト)
スプライトコントローラ	(レジスタすべて)
ビデオコントローラ	(特殊モードを除く全レジスタ)
テキスト	(表示モード，ラスターコピー，同時アクセス)
グラフィック	(表示モード，高速クリア)
パレットレジスタ	(スプライト，グラフィック用)
ジョイスティック	(主要なフォーマットに対応)
本体のスイッチ	(リセット，NMI，POWER)
LED	(本体とキーボード上)
＜実装途中＞	
MIDIボード	
ビデオコントローラの特殊モード	
Windowsのファイルシステムへのアクセス	
＜未実装＞	
プリンタポート	
RS-232C	
ADPCM	(録音，PAN機能)
DMA ch0，ch1	(ハードディスクとフロッピーディスク)
RTC	(時計の書き込み，アラーム)
MFP	(USART送信)
グラフィックVRAMへの画像入力	
フロッピーディスクコントローラ	(IOCS経由のため不要)
ハードディスクコントローラ	(IOCS経由のため不要)

リスト5　キーコードの変換テーブル

```
;================;ATのスキャンコードをX6のスキャンコードに変換する
;ex. X6 scan code = AT scan code
;キー割り込みエミュレーションをチェックした時のコード変換テーブル
;
/key.at2x
70 = 10                     ;=vk_shift
71 = 11                     ;=vk_ctrl
54 = 13                     ;HELP=vk_pause
39 = 21                     ;ROLLDOWN=vk_page up
38 = 22                     ;ROLLUP=vk_next
.
.
.
;==================================================================
;
;ここからはIOCSをフックした時(割り込みを使わない)の変換テーブル
;
;WM_KEYUP,WM_KEYDOWNのスキャンコードからX6のスキャンコードに変換する
/sca.at2x
;左はX68K keyのスキャンコード
;=の右はPC-AT keyのスキャンコード
70 = 10                     ;=vk_shift
71 = 11                     ;=vk_ctrl
54 = 13                     ;HELP=vk_pause
39 = 21                     ;ROLLDOWN=vk_page up
38 = 22                     ;ROLLUP=vk_next
.
.
.
/asc.at2x
10 = 09                     ;TAB
0f = 08                     ;BS=vk_back
1d = 0d                     ;CR=vk_return
01 = 1b                     ;ESC=vk_escape
35 = 20                     ;SPACE=vk_space
02 = 21                     ;!
03 = 22                     ;"
.
.
.
```

中間形式に変換したうえでX68000で使われているなかから，5種類のフォーマットへと変換しています。しかし当初は仕様書にある標準のもののみを実装していましたが，ユーザーの間では拡張されたものも使われていることから寄せられたアドバイスを基に現在の形での実装になっています。

ジョイスティックもまた以下のように変換方法に応じてテーブルを定義しています。

リスト6　ジョイスティックの変換テーブル

```
;PAD出力の定義ファイル
;選択肢 = EXTbytes+STDbyte
; 8421 方向キー
; ABCDEFGHIJKL ボタン定義 並びは左よりボタン1から..
; * は未定義
; (フォーマット) スペースは必ず1つ必要。
/pad.at2x
0 = * B A * 8 4 2 1 * B A * 8 4 2 1              ;標準 2ボタン
1 = * BC AC * 8 4 2 1 * BC AC * 8 4 2 1          ;A+B=C 2->3ボタン
2 = * B A * 8C 4C 2D 1D * B A * 8C 4C 2D 1D ; X=u+d C=l+r 2->4ボタン
3 = * D C * 8 4 2 1 * B A * 8 4 2 1              ;4ボタン
4 = * G C * H D E F * B A * 8 4 2 1 ;8ボタン
5 = * H G * F E D C * B A * 8 4 2 1 ;予備(ユーザー定義用)
```

X68000ではファイルの入出力は，フロッピーディスク，ハードディスクともにIOCSルーチンを経由して読み書きを行います。EX68ではこのIOCSルーチンをフックして，周辺LSIのエミュレーションを使わず，ディスクを単一のイメージファイルとしてアクセスしています。

この方法は安全である半面，直接ハードディスクやフロッピーディスクのコントローラをアクセスするプログラムがうまく動作しない原因にもなっています。

ここでも当初はフロッピーディスクの全データを並べた拡張子XDFのイメージファイルを作成して使用していましたが，実機ではすでにいくつものフォーマットのイメージファイルが存在していましたので，ユーザーのリクエストの中からイメージファイルを小さくできる拡張子DIMのファイルの読み込みに対応しました。

また拡張子XDFとハードディスク用の拡張子HDFのファイルはイメージファイルですので，Windowsからは直接この中のファイルの読み書きができません。これは不便でしたが，これもまた別の作者にツールを作っていただき，実際にディスクを介することなくファイルの転送ができるようになりました。

出力にはサウンドのFM音源，AD PCM，MIDIがあります。AD PCMは圧縮されたデータですので一括して展開し，またサンプリング周波数もPCのサウンドカードとは違いますので周波数を変換したうえでWindowsのWAV形式にして出力します。MIDIはエミュレーションされたMIDIの拡張ボードからメッセージを取り出し，そのままWindowsのMIDIデバイスへと渡しています。

当初，FM音源の再現性はよくないだろうとの予測から内部動作のみを組み込んでいましたが，ゲームにはなくてはならないとの声がありましたので，サウンドブラスタ用にパラメータを変換して近い音が鳴るようにしました。

PC-98x1ユーザーからはPC-9801-86系のFM音源ボードへの対応の声がありましたので，こちらもユーザーからアドバイスをいただき実装しました。現在は別の作者によるFM音源とADPCMをソフトウェア合成するエミュレータも使えるようになり，サウンドの再現性もよくなっています。

図1　ディスクイメージファイルの指定

X68000には本体とキーボードにLEDやスイッチがありますが，PCでは該当するデバイスがありませんので，別のウインドウを作成してまとめて表示します。

画像処理

最後はいちばん負荷がかかりますが，X68000再現のためには重要なビデオ画像の合成があります。ここまでのエミュレーションでEX68内部には独立した合成前のデータが実機のVRAMとほとんど同じ形式で作られていますので，重ねあわせる優先順に従い，下になるプレーンから座標変換とカラーパレットの変換を行いつつメインメモリ上のフレームバッファへ重ね書きしていきます(図2)。

X68000のグラフィック画面は最大で4面表示され，各ピクセル当たり2バイトを使用するビットマップです。テキスト面もまたグラフィック面とは形式の違うビットマップで，ピクセル当たり1ビットの面を4つ重ねて1面を構成します。スプライトはゲームに最適化されたハードウェアで8×8または16×16ドットのキャラクターをタイル状に敷き詰めた面を2つと128または256個のキャラクターを1ドット単位に配置して自由に重ねることができるようになっています。

通常はこうしてフレームバッファに作成されたビデオ画像をWindowsのDIBとしてビデオカード内のビデオメモリへ転送すれば画面表示は完了します。このDIBはWindows95以降では標準で使用できますので，EX68ではパレット付き8ビットカラーとRGBを直接表す16ビットカラーの2種のDIB形式に対応しています。

図2　X68000の表示システムとエミュレートの実際

図3

M68000の動作

画像表示

プロセッサの動作と画像表示は同時に動作している。
走査線にあわせてVRAM，パレットスクロールを書き換えることでラスタスクロール，スプライトダブラを実現する。

X68000実機の１画面の表示期間

仮想のVSYNC

VSYNC

仮想時間

M68000エミュレーション

ダミーの画像表示期間
書き換えの記録を取る

画像合成

ビデオカードへの転送

Windowsの画像表示

プロセッサの動作と画像表示は同時に動作していない。
ダミーの表示期間に記録を取り，画像合成時に再現する

EX68の１画面表示の時間割り当て

図4

M68000
instruction(); 命令解釈処理
interrupt(); 割り込み処理
MPU.cycle.cur++ 仮想時間内でのクロック生成

インタフェイス
read(addrs); 読み出しルーチン
write(addrs,data); データ書き込みルーチン
interrupt(intnum); 割り込み要求処理

周辺LSI群 アドレスにより分岐する
read/write MainMem() メインメモリ
read/write GraphMem() グラフィックメモリ
read/write TextMem() テキストメモリ
read/write SpriteMem() スプライト
read/write PaletteMem() パレット
read/write MFP() マルチファンクションペリフェラル
read/write DMAC() DMAコントローラ
read/write CRTC() CRTコントローラ
read/write FM() ＦＭ音源LSI
read/write ADPCM() ADPCM LSI
read/write RTC() リアルタイムクロック
read/write SCC() シリアルコミュニケーションコントローラ
read/write JOYSTICK() ジョイスティックポート
.
.

Windowsとの入出力
BitBltDib() 合成画像の転送
gen_text() テキスト合成
gen_graph() グラフィック合成
gen_spr() スプライト合成
palette() パレット変換テーブル作成
fmemu() ＦＭ音源変換
set_adpcm() ADPCMをWAVに変換
set_scan_code() キー入力処理
mouse_message() マウス入力処理
chk_joy() ジョイスティック入力処理
.
.

このほかのWindowsのカラーモード32ビットカラーなどへはWindowsが自動的に変換しますが，この場合は相応の処理時間が余分にかかることになります。

このような通常の表示以外に実機では画面の走査線の位置にタイミングをあわせて頻繁にスクロールや表示のオン/オフ，スプライトの書き換えなどを行うことで，ソフトウェアによって，多重スクロールや表示できるスプライト数を増やすなど，さらに高機能を実現しているゲームプログラムがあります。これらは動作原理の違いからこれまでのルーチンでは実現できません。

これらの再現にはリードライトのエミュレーションの段階から仮想の走査線を導入してタイミングを管理し，書き換えたデータを記録しておく必要があります。このタイミングの記録と書き換えたデータの記録を参照して画像合成の際に変換を行いラスタスクロールやスプライトダブラを再現しています(図3)。

このほかにも走査線ごとにカラーパレットを書き換えたり，画面の重ねあわせを入れ替えたりと多くの手法が使われています。EX68ではこれらの再現も不可能ではありませんが，処理時間との兼ねあいから実装していません。

画像合成に負荷がかかる原因は，X68000が最大で８面の重ね合わせを行うことにあります。なかでもテキスト，グラフィック面の描画はデータ量が多いうえに局所性がなく，データキャッシュがヒットしないため処理時間の多くがメモリの読み書きにあてられます。これに対しては処理速度を上げるために書き換えられたところ，ブランクとなるところの記録を利用してできるだけの最適化を行います。

また合成後のVRAMへの転送は一般的にWindowsでは処理が遅いと思われていますが，最適に設定された状態では転送速度に違いはありません。EX68ではDirectXが使えるときは16ビットカラーに切り替え転送速度を計測したうえで最適化しています。

ここまでのエミュレーションの各部の関係とデータの流れは図４のようになります。

エミュレータとしてはできる限り実機の再現を目指すほかに，使いやすくするため，独自の機能拡張も行います。イメージファイル内のファイル以外にWindowsのファイルをそのまま読み書きをするデバイスドライバWINDRV.SYSを導入することでWindowsのネットワークドライブやCDROM，MOなどへもアクセスできるようになります。

これによりWindowsから起動したX68000ウインドウ内でバッチやスクリプト，データを共有できます。

高速化という点でも，X68000のソフトウェアによる浮動小数点演算ドライバへの呼び出しをフックしてPentiumのFPUを使った演算に置き換えることで，これらの演算を多用するソフトでは実機より遥かに高速に動作します。

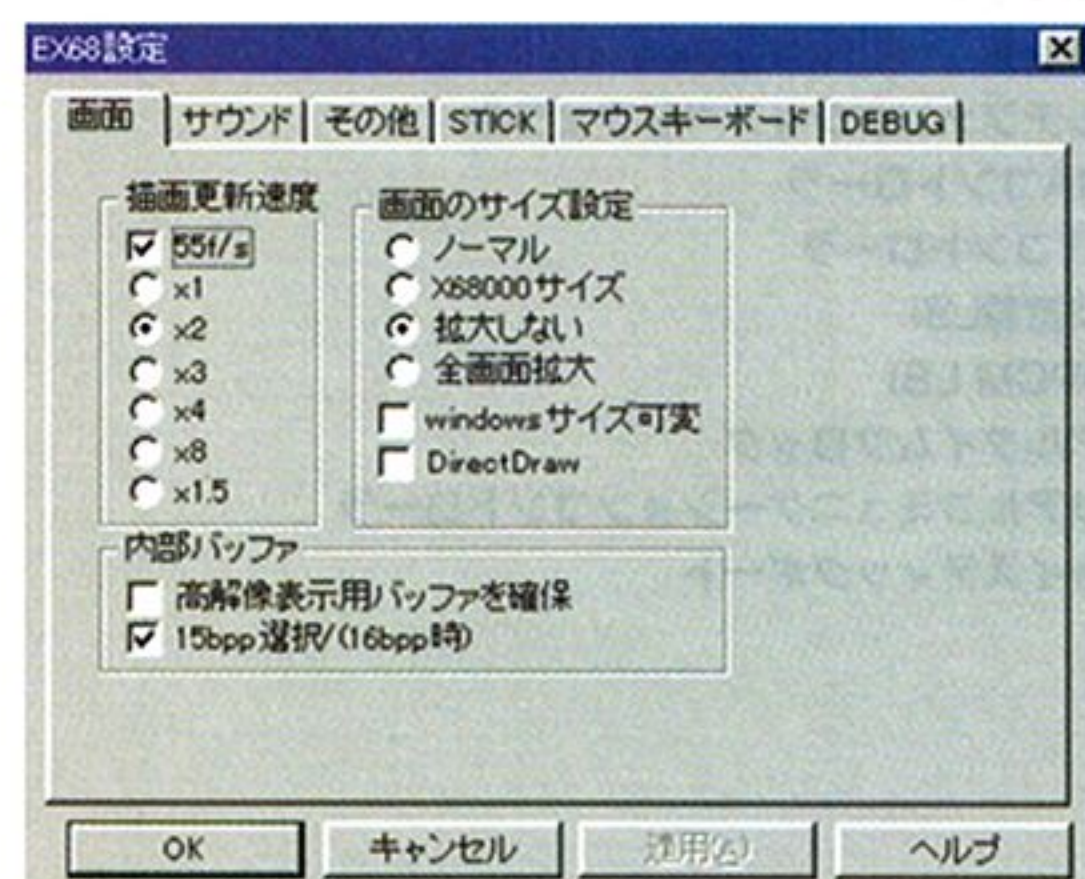
こうしてWindowsの中に再現された，思い入れのあるパソコンの姿を見るのは楽しいものですが，数十倍のCPUパワーを使って初期の16ビットパソコンを再現するという試みは一種のジョークプログラムのようにも思えます。

最後に。このようにまだ不完全なエミュレータですが，掲示版やメールで多くの技術上のアドバイスをいただき，バージョンアップのたびに動作テストにも協力していただいて，少しずつ実機に近づいてきました。皆様にはいつも応援していただき大変感謝しています。

http://www.kumagaya.or.jp/~yamama/emul/

動作設定の意味について

EX68ではメニューのオプション項目を開くことで，さまざまな設定を行うことができる。エミュレータ全体の動作にかかわるものから，グラフィックやサウンド，ジョイスティックといった各個別のハードウェアをどのように処理するかをここでカスタマイズすることでより，快適な動作が可能になる。グラフィック負荷を調整したり，ジョイスティックの設定を行うなどは必須事項だろう。

●画面

ここでは使用しているPCのパワーにあわせてEX68の動作を変えられます。描画更新速度の設定が必要な理由はPCのパワーによってはEX68の画像合成が間にあわないことにあります。このような場合には画像合成を省略し，前回の画像を使うことで，プロセッサエミュレーションの時間を確保します。

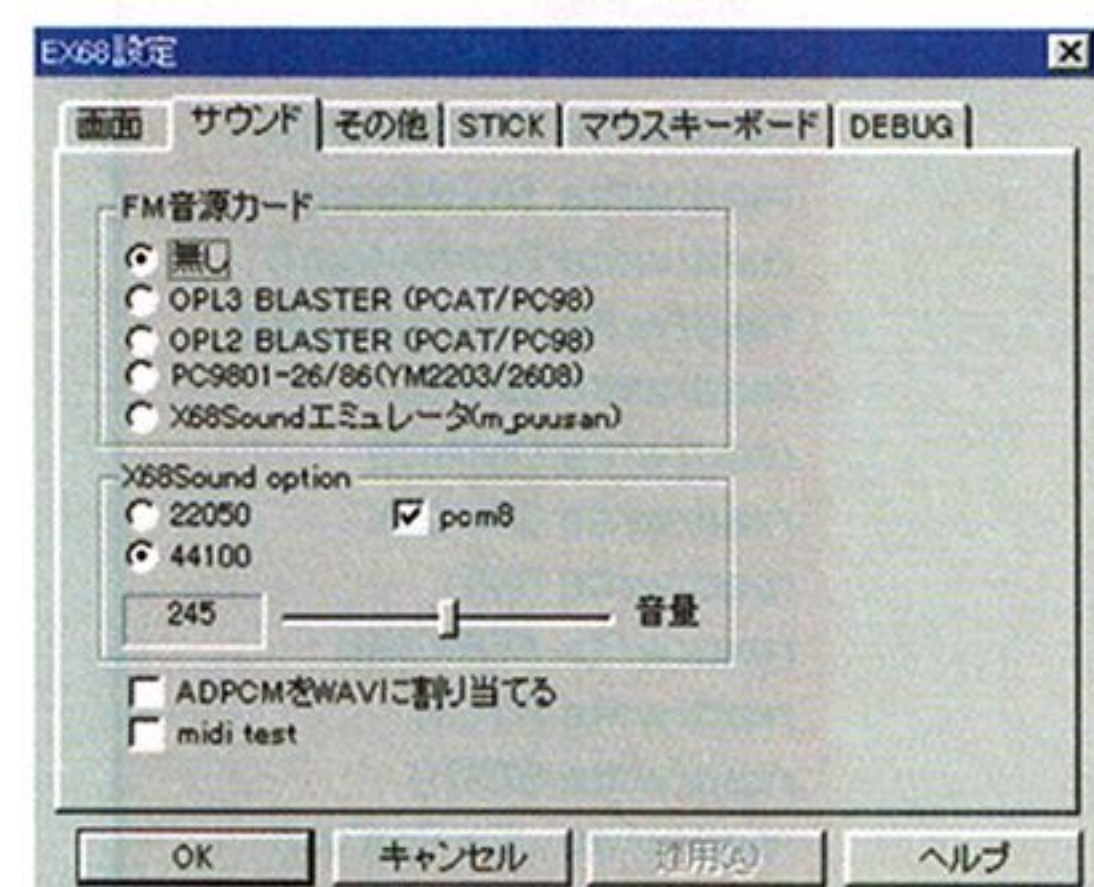
55f/sはゲームなどが速くなったり遅くなったりしないように一定の速度で進行するよう速度を再現するための設定です。仮想の時間を扱うエミュレータに対する外部から実時間へ同期させる処理のひとつです。

画面のサイズ設定は，ビデオカードの描画能力により選択します。拡大表示はビデオカードのチップやドライバにより処理速度が大きく変わりますので最速は"拡大しない"場合ですが，そのほかはノーマルで縦横2倍，X68000サイズは実機と同じ画面の縦横比を，全画面拡大はウインドウいっぱいに拡大する表示になります。

Windowsサイズ可変はウインドウデスクトップ全体の解像度を変更することで，EX68ウインドウの拡大と同じ効果を得られます。DirectDrawが使えるならさらに設定できる解像度が多くなっています。

内部バッファはX68000で使用されているSX-WINDOWやKO-WINDOWの高解像度表示を使うときに内部バッファを多く確保するための設定です。これはX68000のハードウェア仕様書にもありませんが，実機では使えるようになっています。

15bpp選択/(16bpp時)は速度を計測したうえで自動的に設定されますが，これはビデオカードによって15ビットカラーと16ビットカラーのDIB描画処理速度に違いがあるためで，速いほうを選択しています。

●サウンド

ここではサウンドエミュレーションの動作を選択できます。

FM音源カードではSoundBlaster 2種またはPC-98x1系のFM音源カードを使ったパラメータ変換によるFM音源のエミュレーション，X68Soundエミュレータを使用したソフトウェアによるWAVへのFM合成を選択できます。

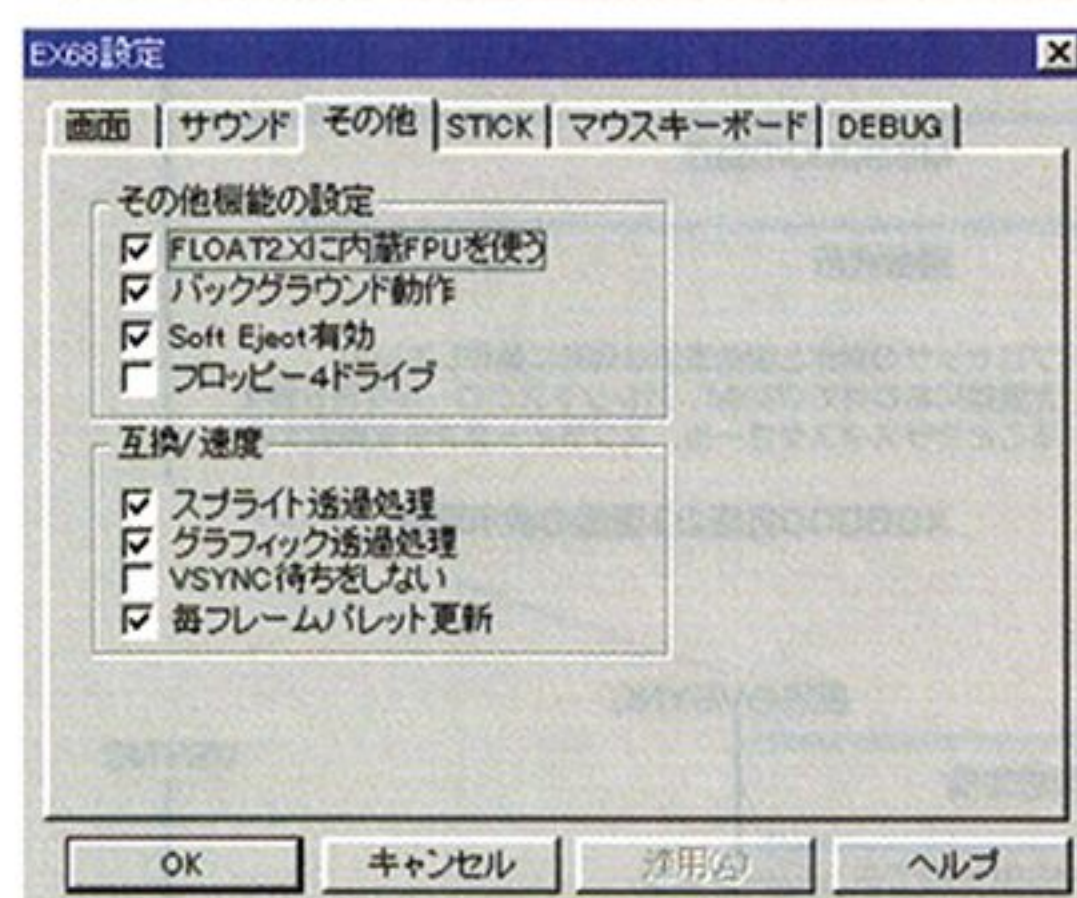
ただしFM音源カードへの直接アクセスができるのはWindows95/98のみです。WindowsNTでは通常このようなIOポートは保護されていますので，X68Soundエミュレータを選択することになります。

X68SoundエミュレータではFM音源の合成のほか，サンプリング周波数の選択，X68000でのソフトウェアによるPCMの8音合成をエミュレートする機能もまたPCの能力により選択します。

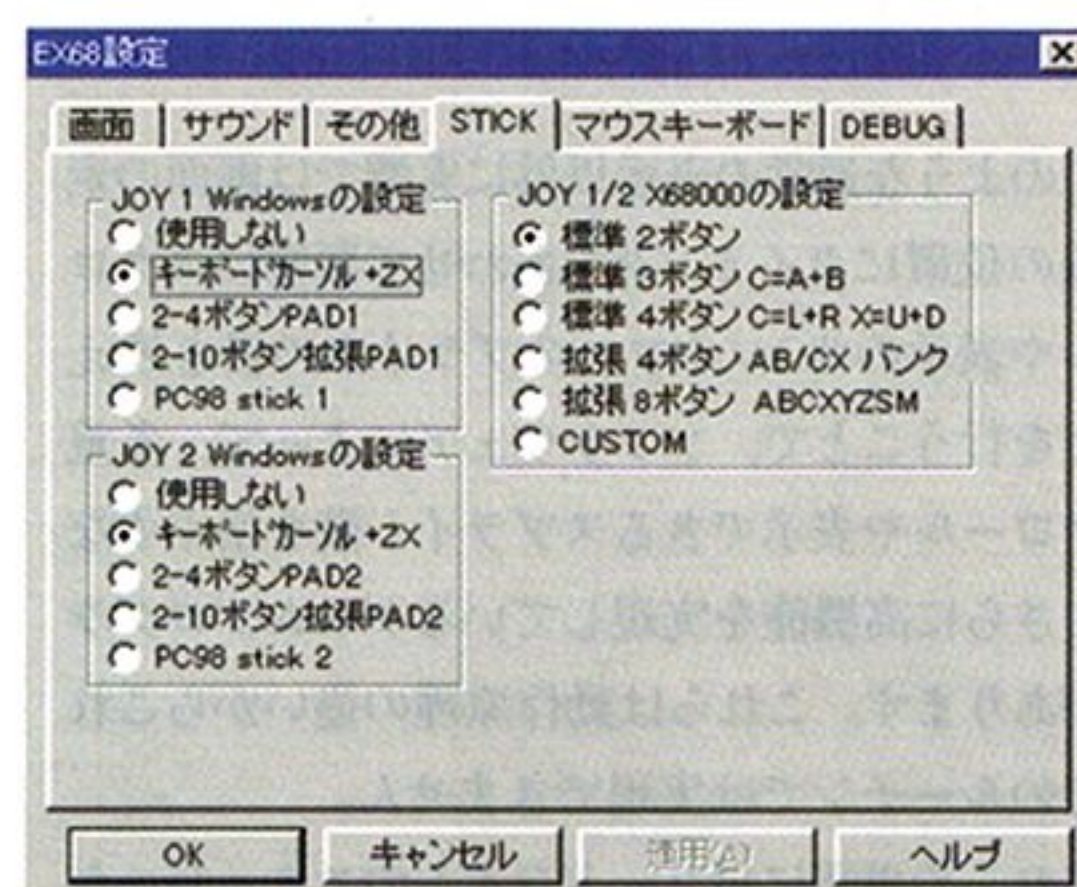
midi testはX68000の拡張ボードをエミュレートする機能で，WindowsのMIDIポートへ変換します。実際にMIDI音源を繋いでいれば負荷は軽くなりますが，ソフトウェア合成のMIDI音源の場合はCPUパワーがないとかなり負荷がかかります。

●その他

ここではエミュレーションの処理に関連して利便性や処理速度を上げるためのいくつかの設定を行います。

FLOAT2.XはX68000の浮動小数点計算をソフトウェアにより計算するデバイスドライバです。EX68ではこのドライバの大部分をPentiumのFPUを使うことで，高速に処理します。

バッググラウンド動作は，EX68がフォーカスを失っていても動作させるオプションです。環境ソフトとして動作させるときなどに利用します。

互換/速度では，ゲームによってはこれらの機能が必要な場合でも，そのエミュレーションを停止させることで処理速度を向上させます。透過処理やパレットを主に画像合成のときに処理する特殊機能です。

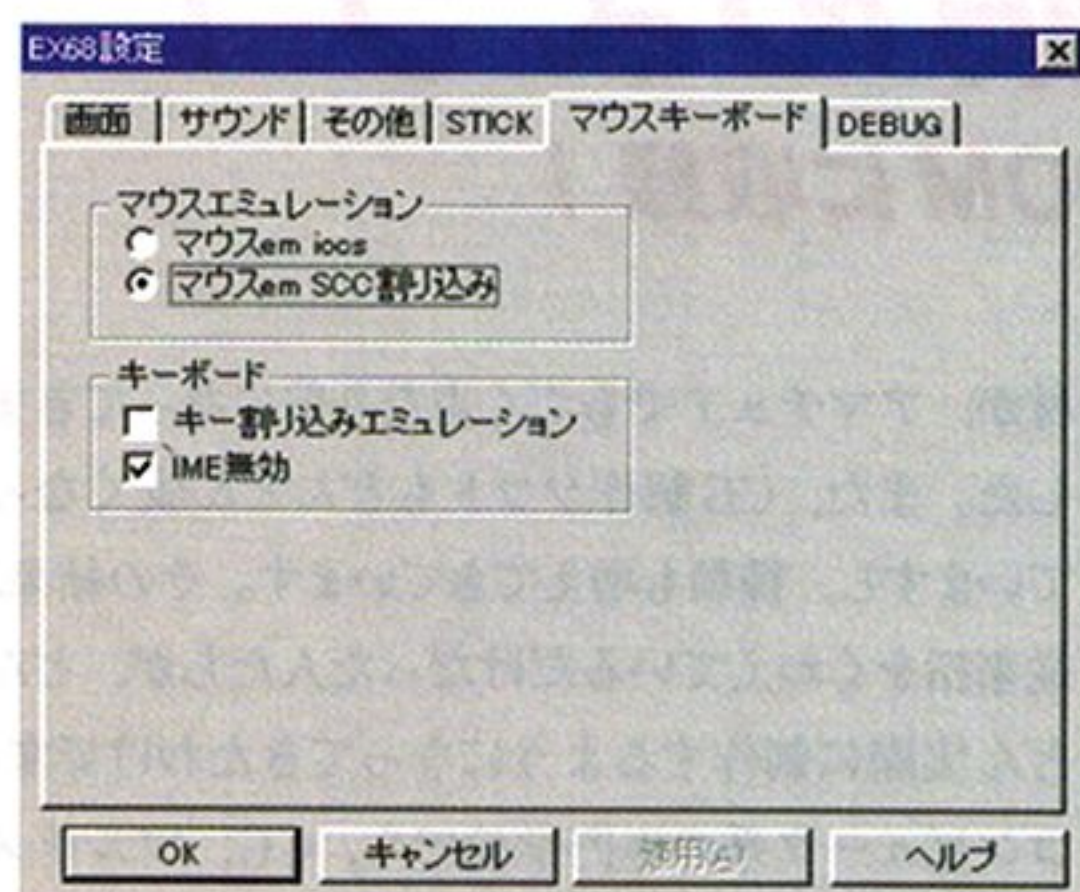
●STICK

Windowsに接続されたジョイスティックとX68000のジョイスティックへの変換を指定します。

Windowsのドライバにより標準化されているジョイスティックと，キーボードに割り当てたエミュレーション，そしてPC-98x1用で使われているデジタルスティックから入力を選択し，出力からはX68000で使われているなかの5種類のフォーマットへの変換を選択します。ここでの変換ルールは定義ファイルAT2X.KEYで設定していますので，ファイルを書き換えることで，独自のルールも設定できます。

●マウスキーボード

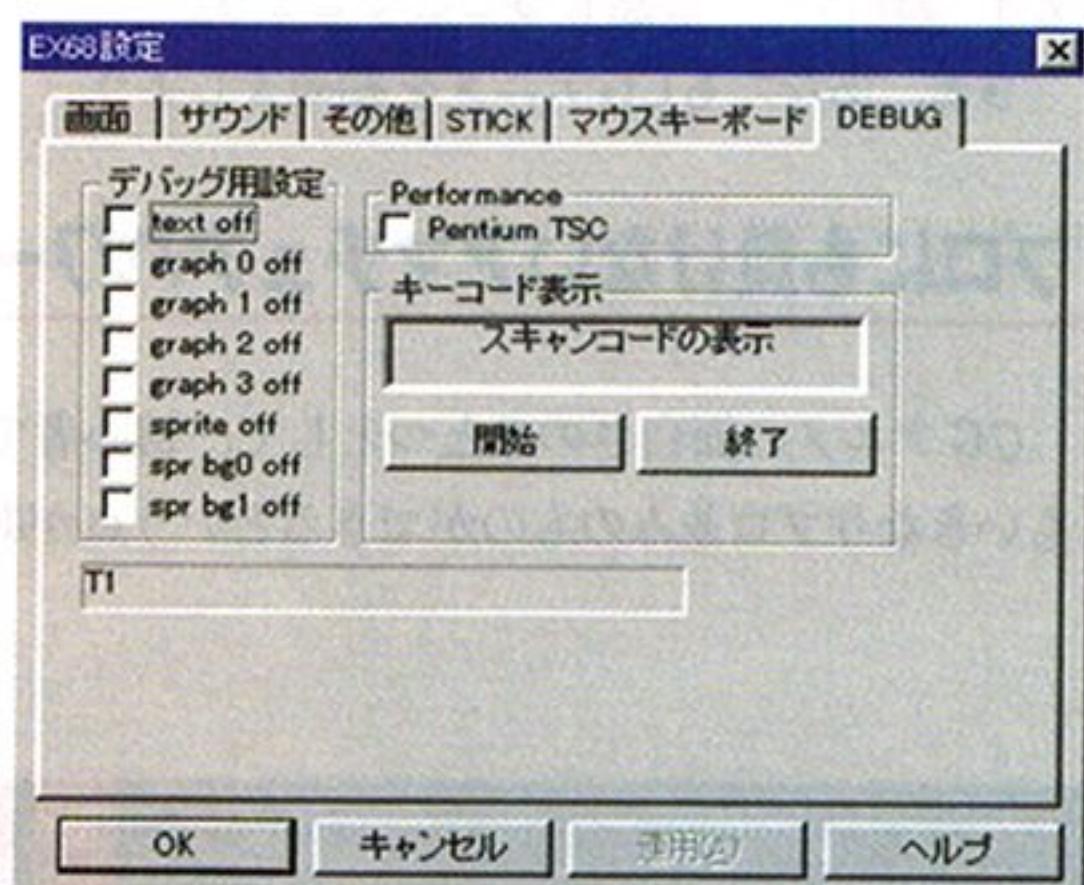
マウスのエミュレーションでは2種類の方法で対応します。

ハードウェアの通信LSIであるSCCへの接続と，ROM内のソフトウェアIOCS呼び出しルーチンへのパラメータ渡しとから，互換性と利便性で選択し

EX68設定
画面 | サウンド | その他 | STICK | マウスキーボード | DEBUG
マウスエミュレーション
マウスem iocs
マウスem SCC割り込み
キーボード
キー割り込みエミュレーション
IME無効
OK キャンセル 適用(A) ヘルプ

ます。IOCS経由の場合はX68000のマウスカーソルとWindowsのマウスカーソルの位置を一致させることができます。逆にSCCへの接続では別々のマウスカーソルが表示される代わりに，X68000としてはより互換性の高い動作が行えます。

キーボードのエミュレーションも同様にLSIへ接続するキー割り込みエミュレーションとIOCS呼び出しルーチンへのパラメータ渡しとから，選択します。キー割り込みを使用するほうが互換性は高くなりますが，キー割り当ての自由度はIOCS経由のほうが大きくなります。IOCS経由ならばキートップのキーシフトも別々に割り当てることができます。

●DEBUG

EX68設定
画面 | サウンド | その他 | STICK | マウスキーボード | DEBUG
デバッグ用設定
text off
graph 0 off
graph 1 off
graph 2 off
graph 3 off
sprite off
spr bg0 off
spr bg1 off
Performance
Pentium TSC
キーコード表示
スキャンコードの表示
開始 終了
T1
OK キャンセル 適用(A) ヘルプ

デバッグ用設定では動作不良時の原因をチェックするために画像合成を部分的に停止させます。

Pentium TSCはPentuim組み込み機能のカウンタを使用して各エミュレーションの処理時間をEX68Windows内に表示させます。

4.9mS 208f/s BitBlt .....画像合成バッファからビデオカードへの転送時間

6.6mS 151f/s V.Gen....画像合成の処理時間

9.5mS MPU.Em.....プロセッサエミュレーションと周辺LSIの処理時間

キーコード表示ではキーボードの変換テーブルを作成するためのスキャンコードの調査に使用します。開始をクリックしてキーボードから入力したコードをAT2X.KEYに定義します。

エラーメッセージと対策の一覧

ディスクから起動できません。
正しいディスクをセットしてください。

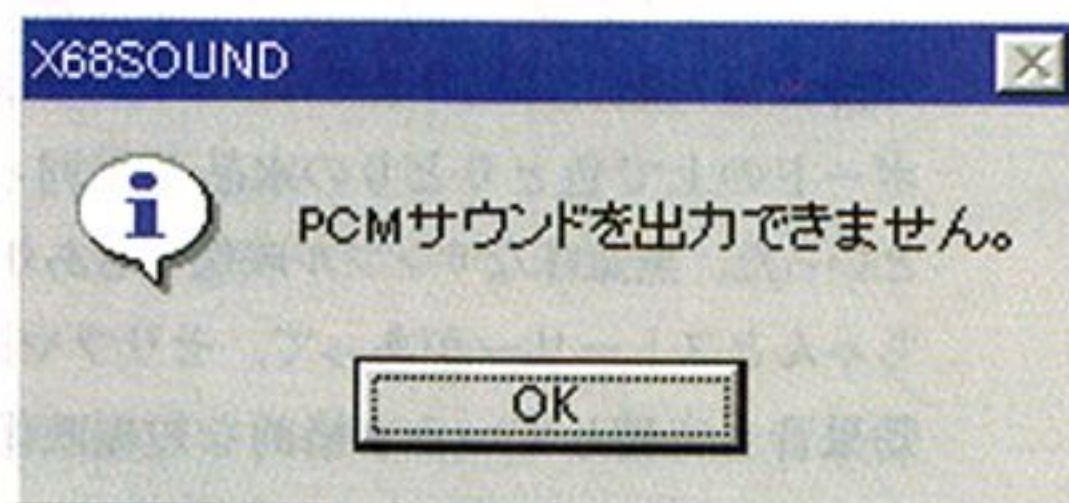
―Human68kを起動できないときのメッセージでIOCSによって表示されています。

対策：EX68は正常に動作していますので，Human68Kの起動ディスクのイメージファイルをX68000ダイアログで指定します。実機ではフロッピーディスクの挿入に相当します。

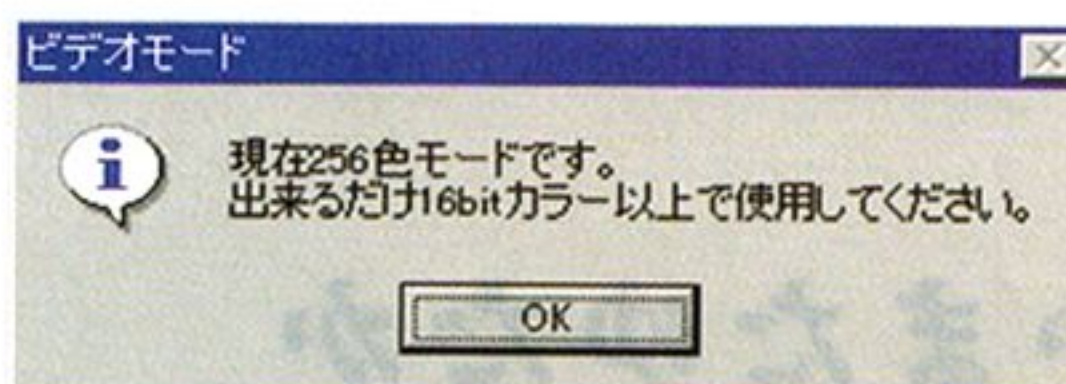
EX68 loading IPL_CG..
C:¥WINPROD¥ex68¥ipl_cg1.datがみつかりませんでした。
OK

―IOCS ROMとCG ROMのイメージファイルが指定されていないとき。

対策：EX68を初めて起動したときのインストールメッセージです。実機でreadipl.xを実行し，作成されたイメージファイルipl_cg.datをEX68.EXEと同じディレクトリにコピーして，ディレクトリを指定します。

X68SOUND
PCMサウンドを出力できません。
OK

―PCMを使うプログラムがほかに起動しているか，サウンドのボードが正常に使用できないときのメッセージです。

対策：PCMを使用しているほかのプログラムを終了します。マルチメディアのプロパティでPCMが使用できるか確認してください。

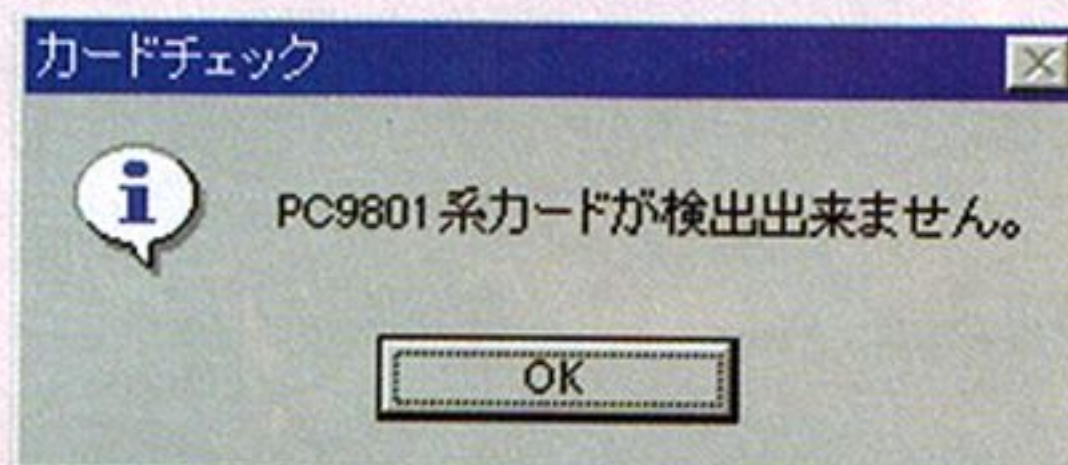
ビデオモード
現在256色モードです。
出来るだけ16bitカラー以上で使用してください。
OK

―画面が8ビットカラーで動作しているときの警告です。EX68は16ビットカラーに最適化されています。

対策：画面のプロパティを開きディスプレイの詳細のカラーパレットでHigh Color(16ビット)を選択してください。

環境変数
環境変数 BLASTER が設定されていません。
OK

―FM音源のエミュレーションにサウンドブラスタを選択したときにインストール時の環境変数が読み出せないときの警告です。

対策：DOSプロンプトからsetコマンドでBLASTER変数が正しく設定されているか確認します。設定されていないときはサウンドブラスタの再インストールを行ってください。

カードチェック
PC9801系カードが検出出来ません。
OK

―FM音源のエミュレーションにPC-98x1系のサウンドカードを選択し，ボードの検出ができないときの警告です。

対策：ポートアドレスを0x88，0x188，0x288のいずれかに設定します。

Project team DōGA

プロジェクトチーム ドーガ

CGアニメを始めよう！

CGアニメ制作ソフト CD-ROMに収録！

かまたゆたか

パソコンあるならCGアニメ

●いまなぜCGアニメ？

CGアニメ，つまりComputer Graphicsによる映像作品は，現在身の回りにあふれています。テレビ番組のタイトルや映画の特殊効果，またゲームのオープニングなどにもフルCGが盛んに用いられています。ただ，これらのCGアニメは，すべてプロが作ったものです。アマチュアが作ったフルCGアニメなんて見たことがあります？実はこれがいま結構凄い状態なのです。

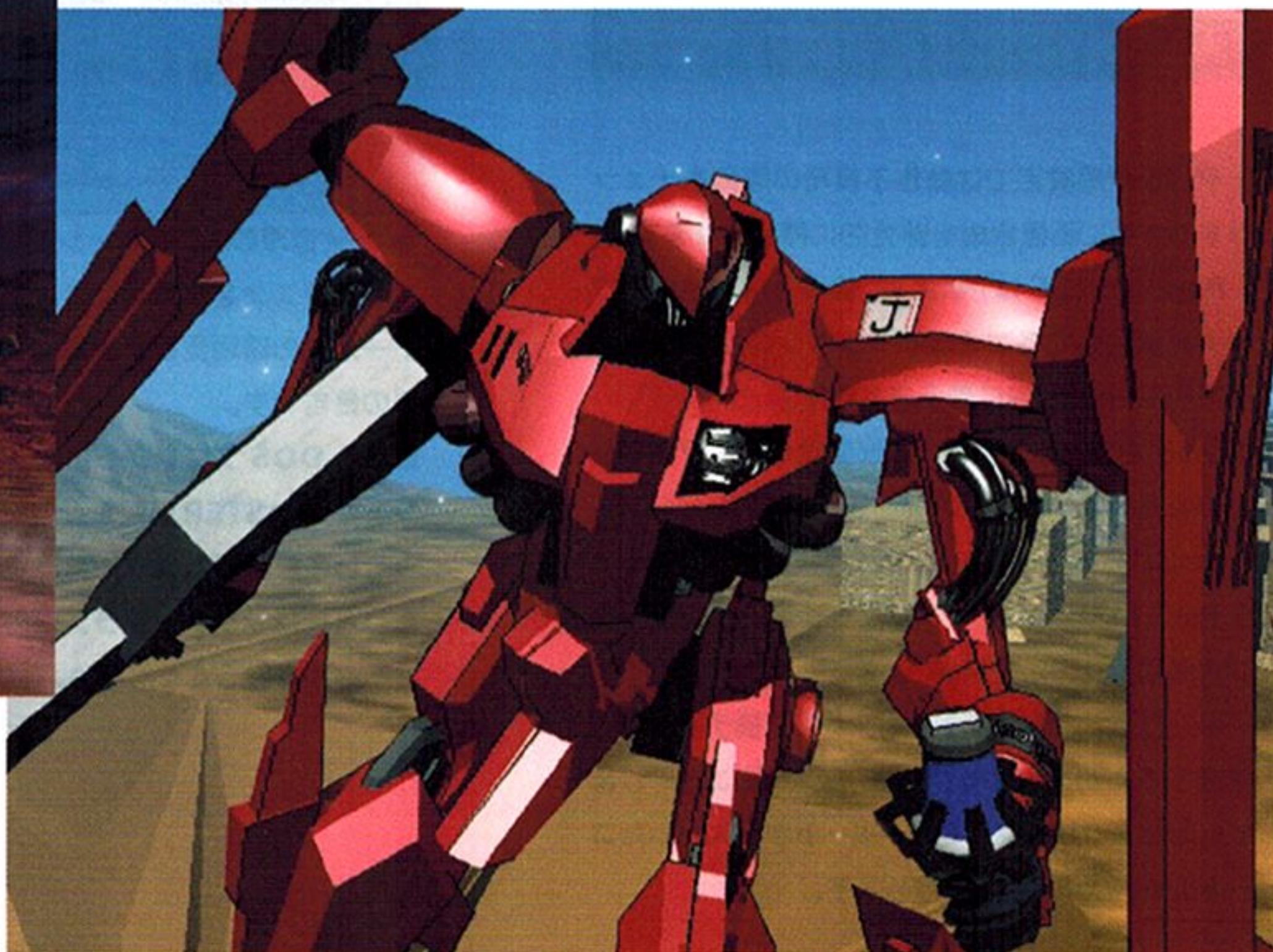
アマチュアの作品といっても，CGアニメというぐらいですから，ペイントソフトで描いた1枚の静止画の類ではありません。また，チェッカーボードの上で色とりどりの水晶玉が回っているといった，無意味なサンプル映像でもありません。ちゃんとストーリーがあって，セリフやBGMや効果音も完備している本格的な短編映像作品です。ギャグもあればシリアスもある，ＳＦやファンタジーもあれば，抽象アート作品もあります。レベルの高い作品が次々に発表され，それらが新しい映像文化を形成しつつあるのです。

その背景としては，まずハードの急速な発展があります。元祖「Oh!X」が扱っていた往年の名機X68000と比較すると，最近のパソコンのCPUパワーは200倍ぐらい上がっています。つまり数年前までCGプロダクションで使っていたような機材が，アマチュアでもごく当たり前になってきました。また，CG制作ソフトもどんどん安くなっていますし，種類も増えてきています。その結果，従来指をくわえているだけだった人たちが，どんどん実際に制作するようになってきたわけです。コンピュータ系の専門学校でも，CG，マルチメディア系のコースは大人気ですし，CG系の専門雑誌も昨年から5，6誌創刊されています。

なぜこんなにCGアニメは人気があるのでしょう？　CGアニメには，きわめて純粋な〝面白さ〟があるのです。皆さんは学生時代，つまらない授業の合間，教科書の隅にパラパラ漫画を作ったことはありませんか？　自分の作ったキャラクターが初めて動き出したとき，思わず〝面白い！〟と感じたことでしょう。この単純な〝面白さ〟は本能に直結した感覚であり，それ以上説明できるものではありません。それがCGアニメの場合，ヘロヘロの線画ではなく，フルカラー，フルモーションで，プロの映像と遜色ないものができるのです。初めてCGアニメを作り，自分のキャラクターが画面を飛び回ったとき，あなたは必ず〝面白い！〟と感じることでしょう。

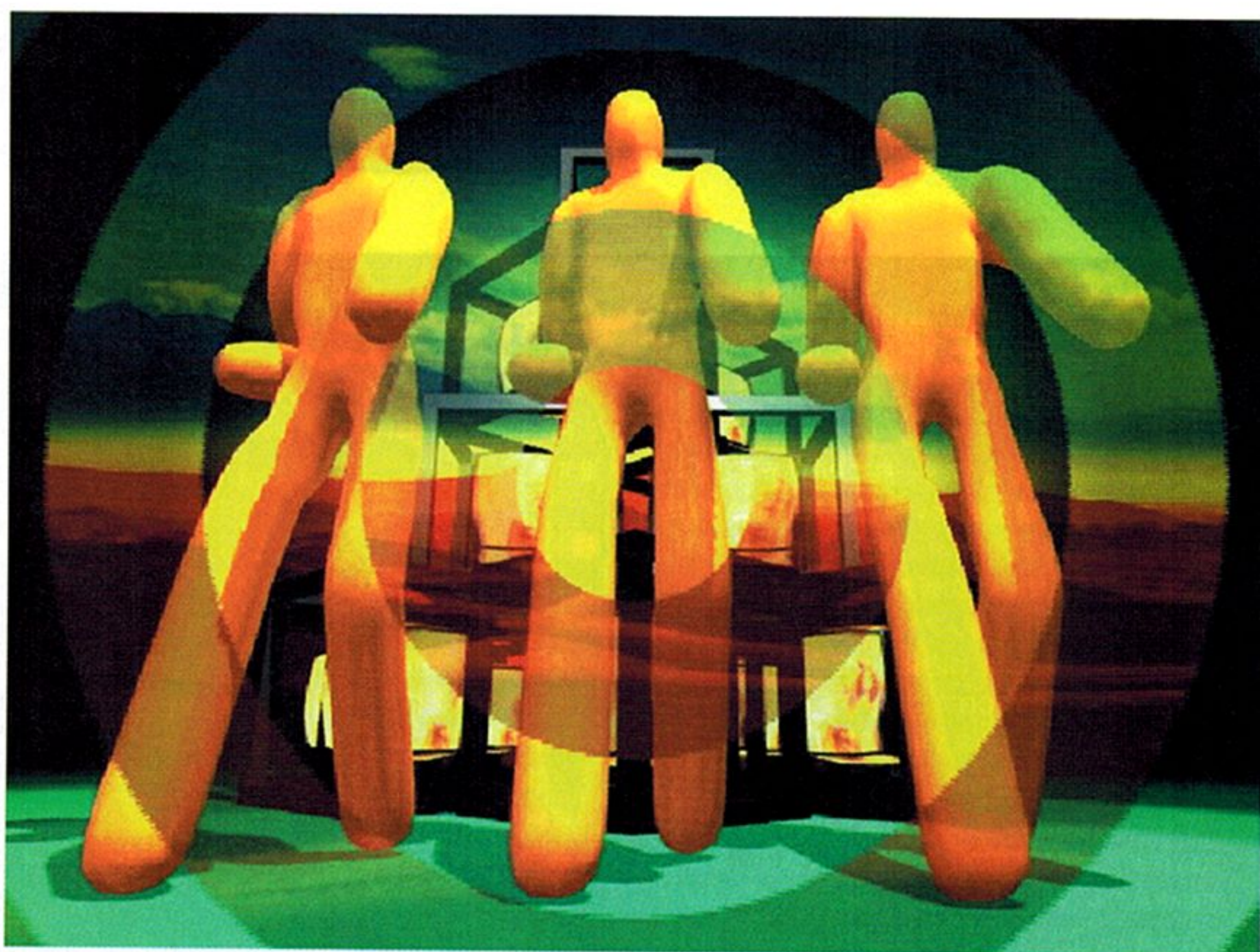
さぁ，あなたもCGアニメを始めましょう。

プロにも負けないアマチュアパワー

CGアニメの面白さのひとつとして，初心者でもいきなりプロ並みのものができるという点があ

ります。なぜなら，CGアニメは，漫画やイラストや絵画と異なり，絵自体は自分で描く必要はありません。プログラムが自動的に描いてくれるからです。そして，使っているハードやソフトも，あまりプロと変わらないものが用意できます。

だいたいプロといっても恐るるに足りません。CGの歴史なんて非常に浅いものですから，その業界のプロといっても，10年以上のキャリアを持つ人なんてほとんどいません。大部分は，CGの専門学校を出たばかりのアマチュアに毛が生えたような人が多いのです。たとえば，昨年行われた某CGコンテストでは，応募者の大半がプロであったにもかかわらず，グランプリ，準グランプリを受賞したのはともにアマチュアだったなんてこともあるくらいです。

それに，アマチュアのほうが有利な面もたくさんあります。まずは時間です。プロの場合，納期という時間的制約がありますから，とにかく間にあわせるために，どんどん妥協しなければいけません。しかし，アマチュアの場合特に締め切りがありません。自分なりに納得がいくまで作り込むことができます。

第一，プロの場合，作る内容を自分で決めることすらできません。クライアントの要望したものしか作れないのです。TV番組のタイトルを作ったり，映画の中の1カットを作るのに，自分なりのテーマやストーリーを盛り込む余地はありません。極端な場合，絵コンテまで決まっていて，自分の好みに関係なく，ただ作業をしているだけというケースも少なくないのです。その点アマチュアは，最初から最後まで，全部自分で決めることができ，自由度がずっと高いといえます。その結果，作品に対する想い入れも違ってきます。アマチュアの作品といえども，プロより面白いものが出てきても，ある意味当然といえるかもしれません。

CGアニメはパソコンのフロンティア

CGアニメは，実写映画やセルアニメと同じ動画映像の一種と考えられていますが，最近私は，むしろ漫画に近いような気がしてきました。なぜなら，実写映画やセルアニメが非常に多くの人たちによる集団制作であるのに対し，漫画やCGアニメは，基本的にひとりの作家（と若干名の協力者）による個人制作だからです。実写映画には必要なカメラマンやライティング，そして役者まで，CGアニメではすべてひとりで制御します。また，セルアニメで人海戦術を要していた動画作成も，パソコンが自動的にレンダリングしてくれます。ですからCGアニメは，漫画と同じように，人の手を借りずとも，自分ひとりで1本の作品を完成することができるのです。

考えてみると，これは，かなり革命的な意味を持ちます。従来映像は，プロが制作したものがテレビやビデオを通じて上映され，一般の人はただ一方的に見るだけでした。しかしこれからは，より多くの方々が個人ベースで映像を制作し，発信するようになるのです。いわば〝映像の民主革命〟ともいえる出来事です。

パソコンでCGアニメができるというのは周知の事実ですが，これが映像の民主革命という一大転機であるということは，まだ一般的に認識されていません。まさにこれからブレイクするところです。まだまだ予想できないような世界が広がっているのです。

ということで，このすばらしい世界をいち早く読者の皆さんに体験していただきたいと思います。

Project team DoGAとは

Project team DoGAは，CGアニメによる新しい映像文化の啓蒙活動を行っている非営利団体。1986年に，大阪大学コンピュータクラブや京大マイコンクラブなどの共同研究プロジェクトとして発足。以後，パソコン用CGアニメ制作ソフトの開発，CGアニメコンテストの主催といった活動を行っている。元X68000ユーザーが多く，元祖「Oh!X」においてもCGに関する長期連載を行っていた。

1993年には，子会社として株式会社ドーガを設立し，CGプロダクション的な活動も行っている(TV東京系「ロスト・ユニバース」など)。

DoGA主催アマチュアCGアニメーションコンテスト作品募集！

このコンテストは「アマチュアのCGアニメーション作品の発表の場を設け，広く一般にPRするとともに，その質的向上を促進する」という目的で，1989年から毎年行われています。

特定のソフトやハードの販売促進を目的としたコンテストとは異なり，使用ソフト，ハードは問いません。また，ほかのコンテストに応募した作品でも結構です。つまり，アマチュアのCGアニメーション作家が自分たちの作品を持ち寄り，今年1年の活動成果を，みんなで見ようという色彩が強くなっています。

ですから，このコンテストに入選すれば，アマチュアCGA作家として一人前などといわれています。しかし，応募作品のレベルは非常に高く，最近は入選するのもかなり難しいといえるでしょう。特に，上位入賞作品ともなると，プロの間からも注目されるぐらいです。

毎年12月31日が締め切りで，3月から4月にかけて，入選作品の発表・上映会が行われています。上映会では，入選作品の上映だけでなく，入選者の座談会，来場者との質疑応答，持ち込み作品の上映なども行われます。また，関連ソフト，ハードの展示会や懇親会などもあり，まさにアマチュアCGアニメーションの祭典といえるでしょう。

この上映会には，千人以上の来場者がありますし，入選作品はビデオ化され，実費で販売されます。ですから，自分の作品をより多くの人に見てもらいたいと思っている方にはうってつけのコンテストです。プロへの第1ステップとして，あるいはアマチュアCG界へのデビューとして，ぜひこのコンテストへご応募ください。

PROJECT TEAM DoGA主催　第11回
Amateur Computer Graphics Animation Contest

■開催主旨

アマチュアのCGA作品の発表の場を設け，広く一般にPRするとともに，アマチュアCGAの質的向上を促進する。

■応募作品要項

コンピュータを使用したアマチュアのオリジナル映像作品。

- 使用機種・使用ソフトは問いません。
- 実写等が含まれていてもかまいませんが，その部分は審査の対象にはなりません。
- プロの方でも，プライベートの機材で，プライベートに制作したものは問題ありません。
- 過去他のコンテストに応募された作品でも応募できます。

■各賞一覧

- グランプリ：賞状，副賞，賞金20万円
- 準グランプリ：賞状，副賞，賞金15万円
- 入賞：賞状，副賞，賞金10万円
- 佳作：賞状，副賞，賞金5万円
- 入選：賞状，副賞，賞金2万円
- 会場審査特別賞：賞状，副賞，賞金1万円

■応募方法

- コンテスト事務局まで，応募票をご請求ください。応募票は，ホームページからも入手できます。
- 応募票に必要事項を記入の上，作品とともに，コンテスト事務局までお送りください。
- 出品料は不要です。

■応募形態

- ビデオテープ(VHS，S-VHS，8，Hi-8，DVC)
- QuickTimeまたはVideo for WindowsのMovieデータ(FD，MO，CD-R)
- ビデオテープに録画したり，Movieデータを作成する環境がなく，画像データなどで応募する場合には，事前にその旨ご連絡ください。

■締め切り

1998年12月31日(当日消印有効)

■審査員

- 「ASAhIパソコン」編集長　三浦 賢一
- 「日経CG」編集長　田島 進
- 「DOS/V POWER REPORT」編集長　土田 米一
- ファミリー4コマ漫画家　寺島 令子
- アニメ監督　渡部 高志
- PROJECT TEAM DoGA代表　鎌田 優

他(第10回，順不同，敬称略)

■注意事項

- 入選作品は主催者が行なう上映会で使用する他CGAのPRのために複製，配布，放送などを行います。
- 作品の著作権は，制作者に帰属します。
- BGMやデザインなどについて，第三者の著作権などを侵害しないよう，十分にご注意ください。
- 応募作品の返却を希望する場合は，その旨申し出てください。原則として返却いたしません。あらかじめコピーをとってください。
- 応募作品に対する不測の事故などの損傷については責任を負いかねますのでご了承ください。
- ビデオに録画する際は，テープの頭に30秒以上，黒画面やカラーバーなどを収録しておいてください。

■問い合わせ先

〒533 大阪市 東淀川区 淡路 5-17-2-102
PROJECT TEAM DoGA内
「CGAコンテスト事務局」
FAX：06-321-4841
E-mail：contest@doga.higashiyodogawa.osaka.jp
URL：http://www.doga.co.jp/ptdoga/

DOGA-LI編

ステップアップ構想

“さぁCGアニメを始めましょう”とはいっても，まったくの初心者には，いくつかの難関が立ちふさがっています。まず，なんだかんだといっても，CGは難しくて，専門知識をたくさん習得しないといけないというイメージがあります。さらに，パソコンは持っていても，CGソフトは購入しなくてはいけません。安くなってきたとはいえ，まだまだ高価です。そして，なんとかソフトを購入して始めたが，あまりの難しさにすぐ挫折してしまったという話はいまだに尽きません。これでは，気楽にCGアニメに挑戦するというわけにはいきません。

しかしご安心ください。当チームでは数年前からこの問題に取り組んでいます。まず，多くの初心者が恐れる“高度な知識と技術の壁”をなんとかしないといけません。

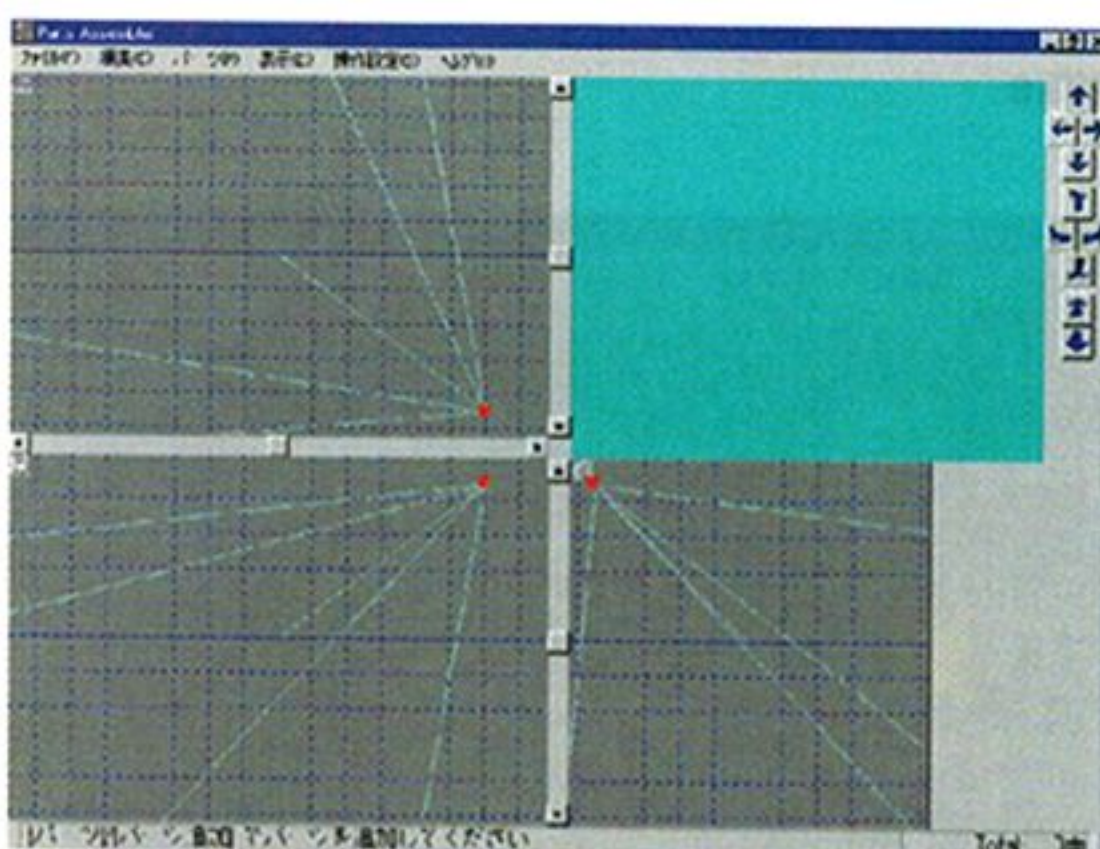
だいたい，ひとつのソフトで，初心者にいきなり基礎から応用まで全部習得しろというのは，高い壁を一気によじ登れというようなものです。そこで，最初は非常にやさしいところから始めて，それを習得してからもう少し難しいものに挑戦するというように，複数のソフトによって，少しずつ段階的に学べるような環境が必要だと考えました。つまり，壁の前に階段をつければ，誰でも越えることができるというわけです。これが「ステップアップ構想」です。

そして，このステップアップ構想に基づいて開発したソフトを，コピーフリー（カンパ希望）ソフトとして，無料配布しています。読者の皆さんも，すぐにCGアニメを始められるように，本誌付録CD-ROMに収録しました。ですからどうやってCGAを始めようなどと悩んでいる暇があれば，このソフトをインストールしてみてください。楽しくCGA作品を制作しながら，CGA制作をひととおり習得できるでしょう。

DOGA-L1とは

DOGA-L1（以下L1）は，このステップアップ構想に基づいて開発した「DOGA-Lシリーズ」の1段目のステップに相当するプログラムです（for Windows95）。

L1では，あまり難しいことは省略して，まずCG制作の流れをひととおりつかむことを目標としています（それでも，結構本格的なCGアニメが作れます）。事前に知っておかなければいけない知識や，覚えないといけない操作を極力なくしているので，本誌を見ながら操作すれば，習得するのに30分とかからないはずです。それでは，

画面1

画面2

順番に解説していきましょう。

画面1がL1のメインメニューです。

「物体をデザインする」

で物体の形を決め，

「モーションをデザインをする」

でその物体がどのように動くか指定し，

「アニメーションを表示する」

で作画し，CGアニメが完成します。これが，CGアニメ制作の流れです。

各々の作業は，このメインメニューから呼び出されるツールで行います。呼び出されるツールは，

「物体をデザインする」：パーツアセンブラ

「モーションをデザインをする」

：モーションエディタ

「アニメーションを表示する」：レンダラー

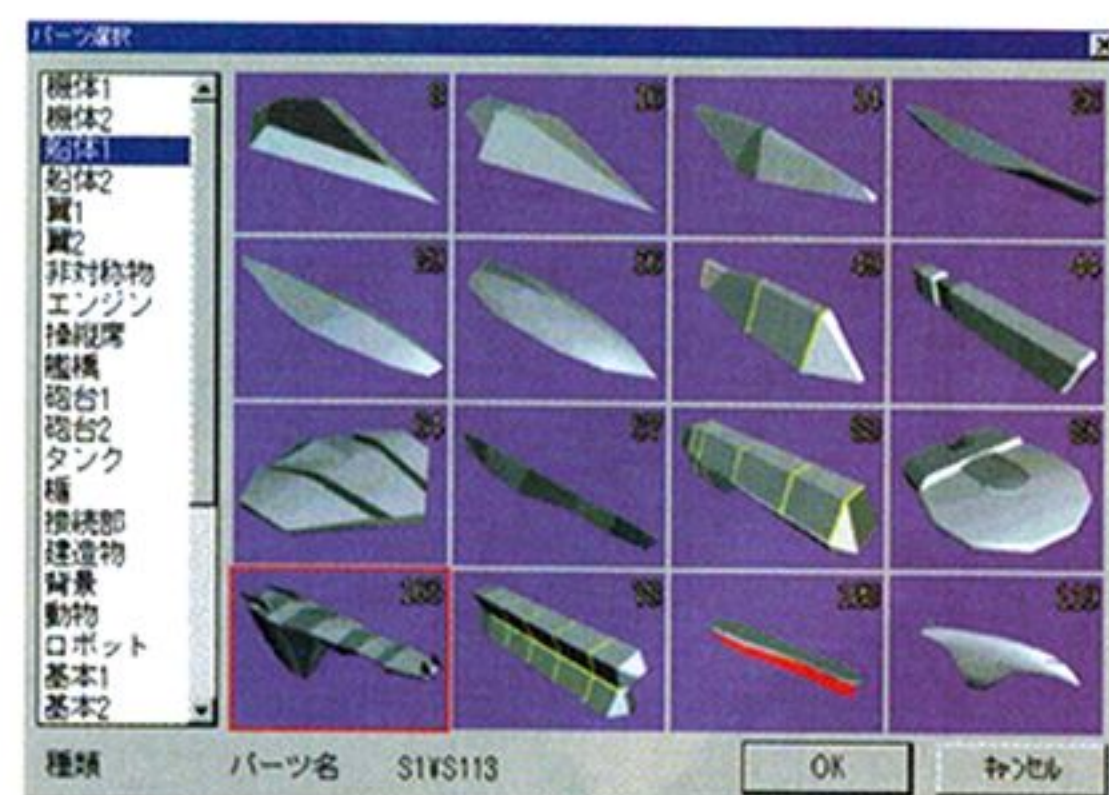
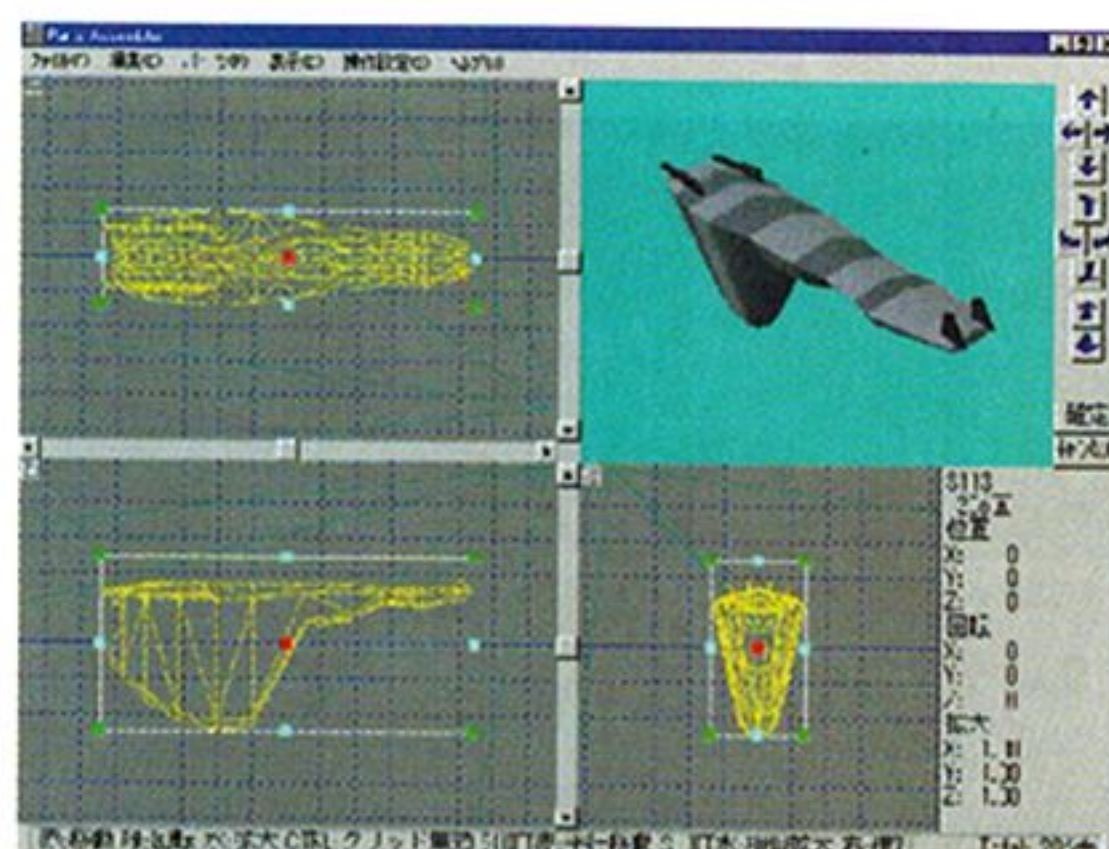
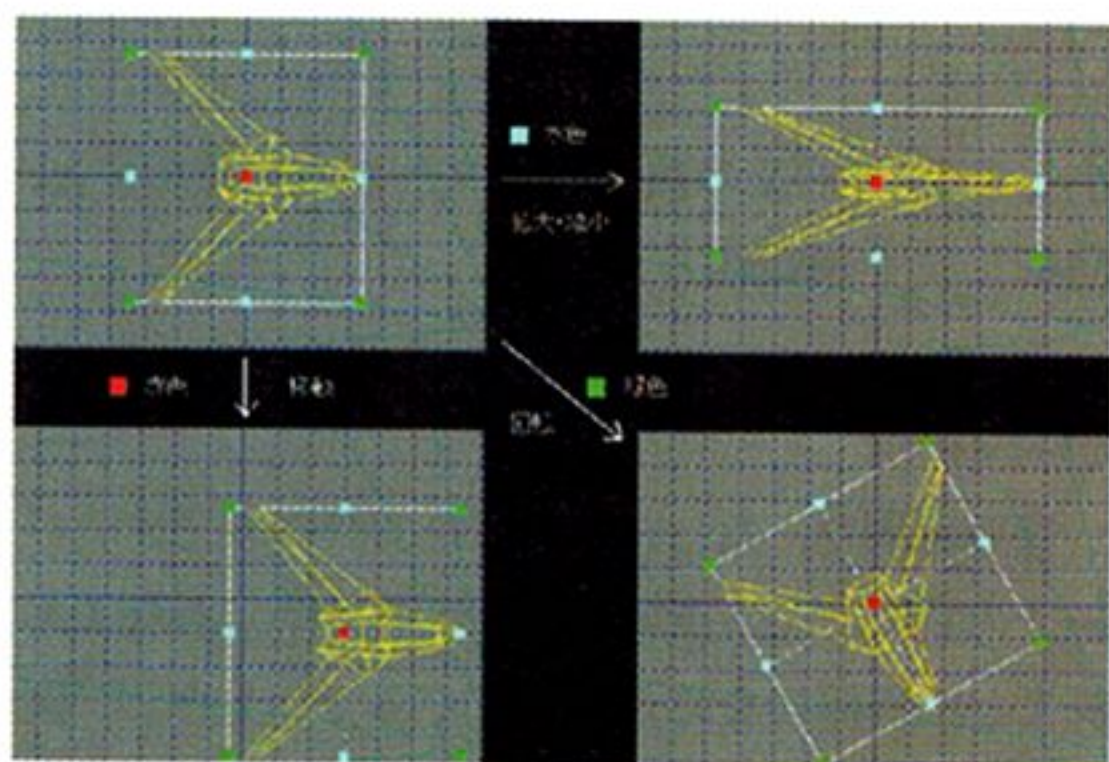
となっています。

L1のパーツアセンブラの使い方

●パーツアセンブラとは

通常モデラといわれるツールの一種で，あらかじめ用意されている300以上のパーツを組み合わせて，ちょうどプラモデルのような感覚で物体を作っていきます。現実のプラモデルとは異なり，どのパーツをいくつでも，任意の位置に置くことができ，組み合わせによってさまざまなものを作ることができます。

●起動画面

画面2がパーツアセンブラの起動画面です。なんか，結構本格的で難しそうですね。第一，これからなにをしたらよいのか，ぜんぜんわかりません。でも大丈夫。そういうときは，画面のいちばん下のメッセージを見てください。常に，次になにをするのか，なにができるかを表示しています。「パーツ」/「パーツ追加」でパーツを追加してください。

画面3

画面4

画面5

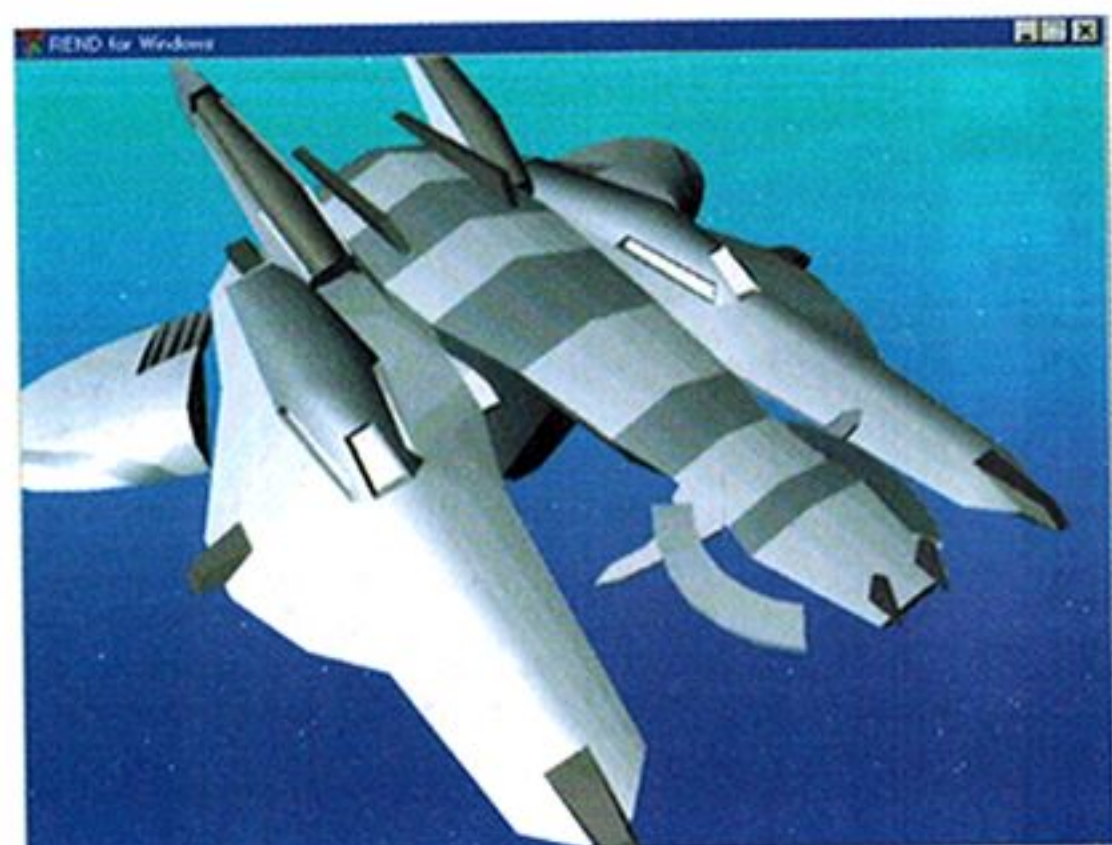
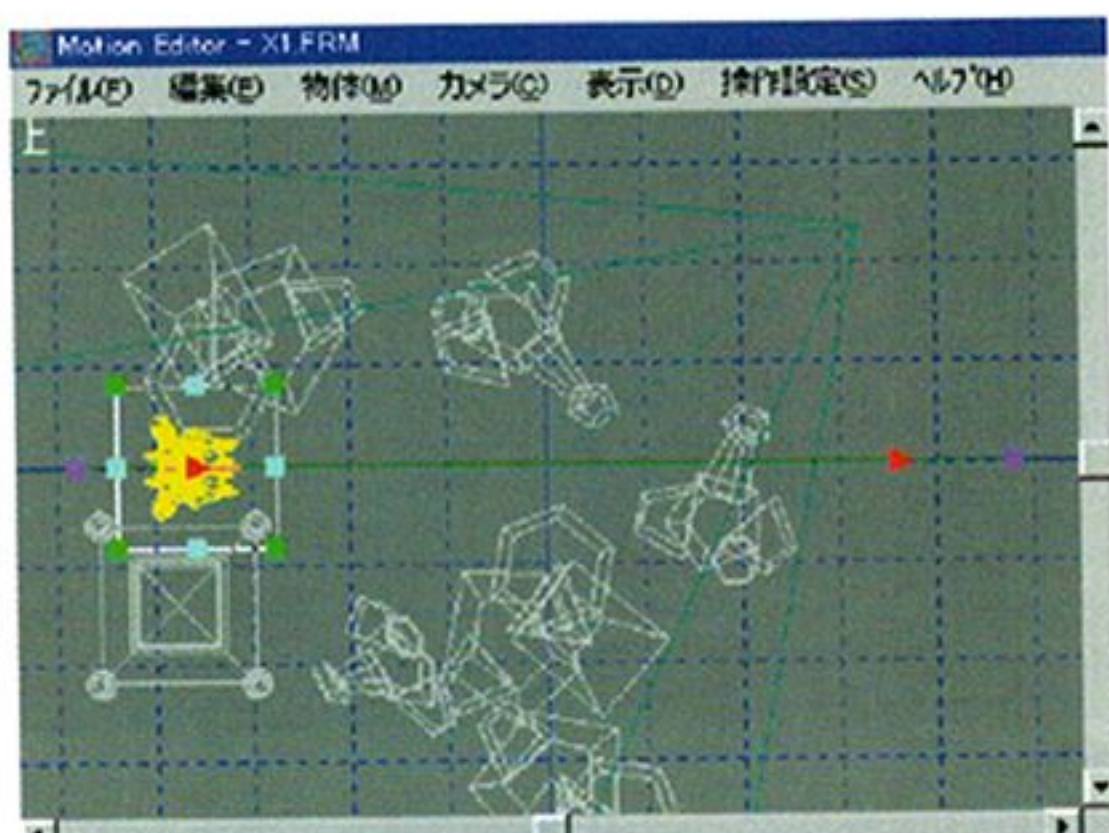
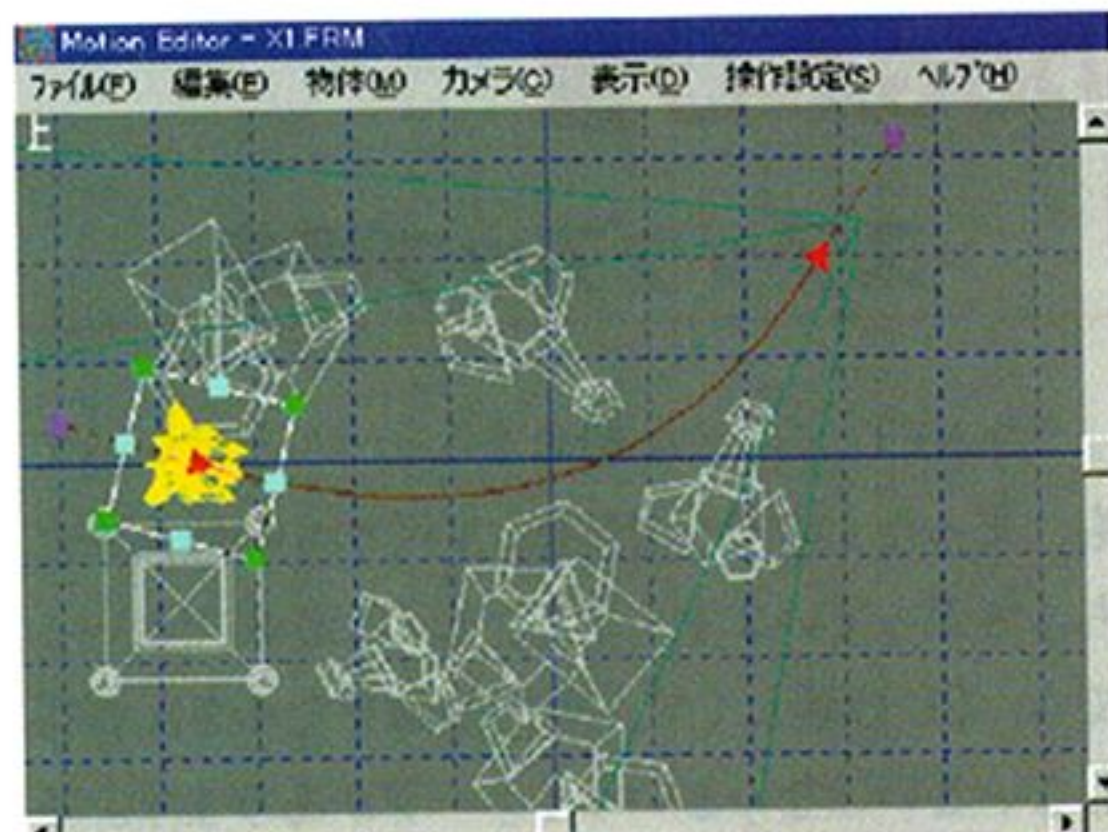
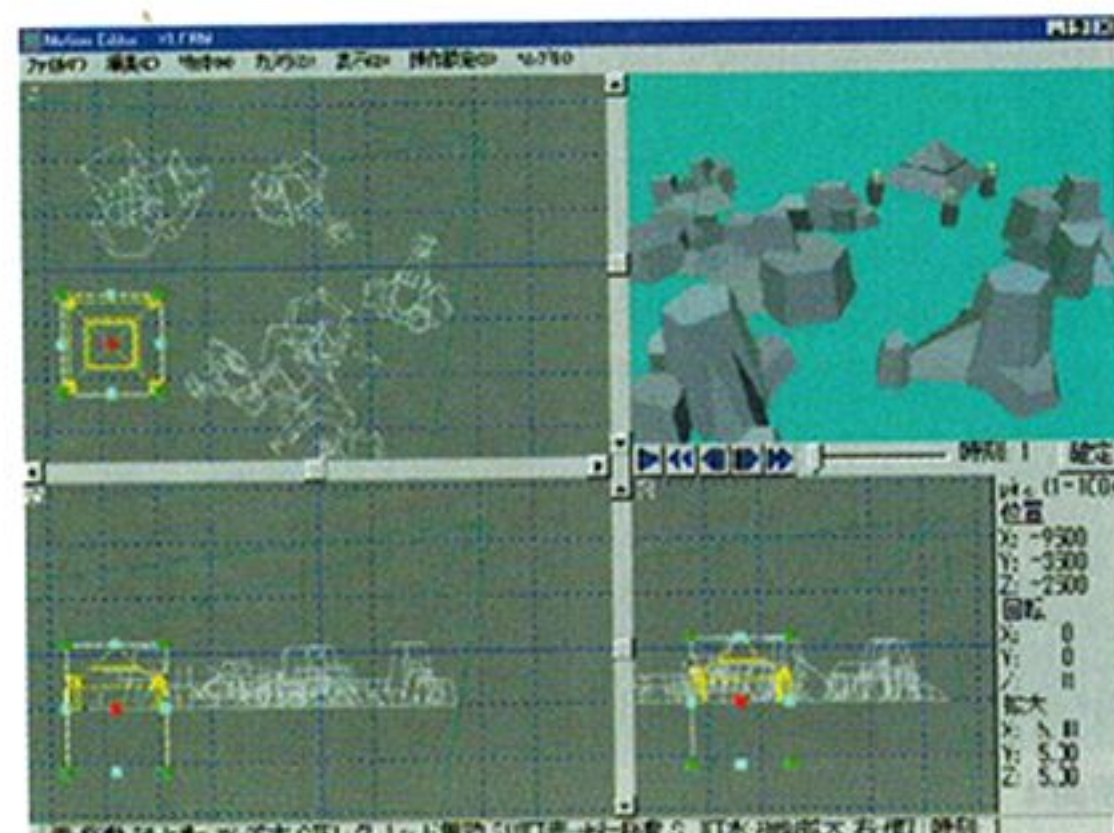
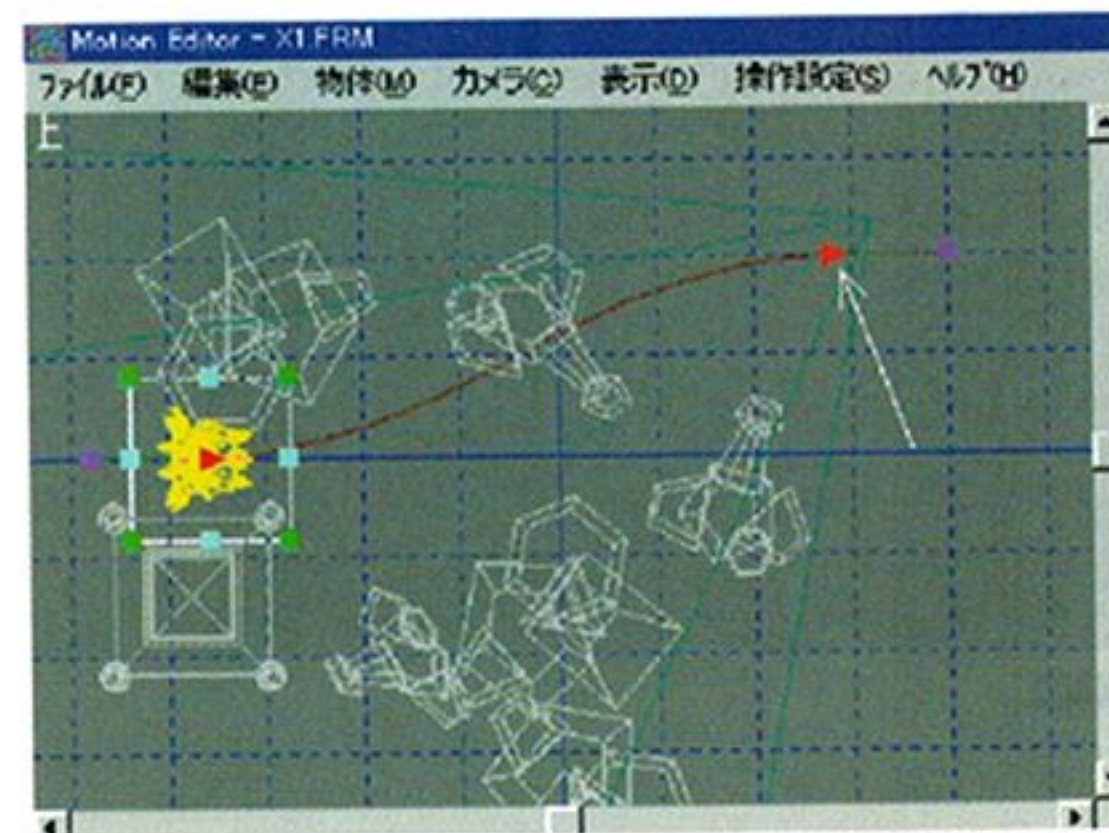
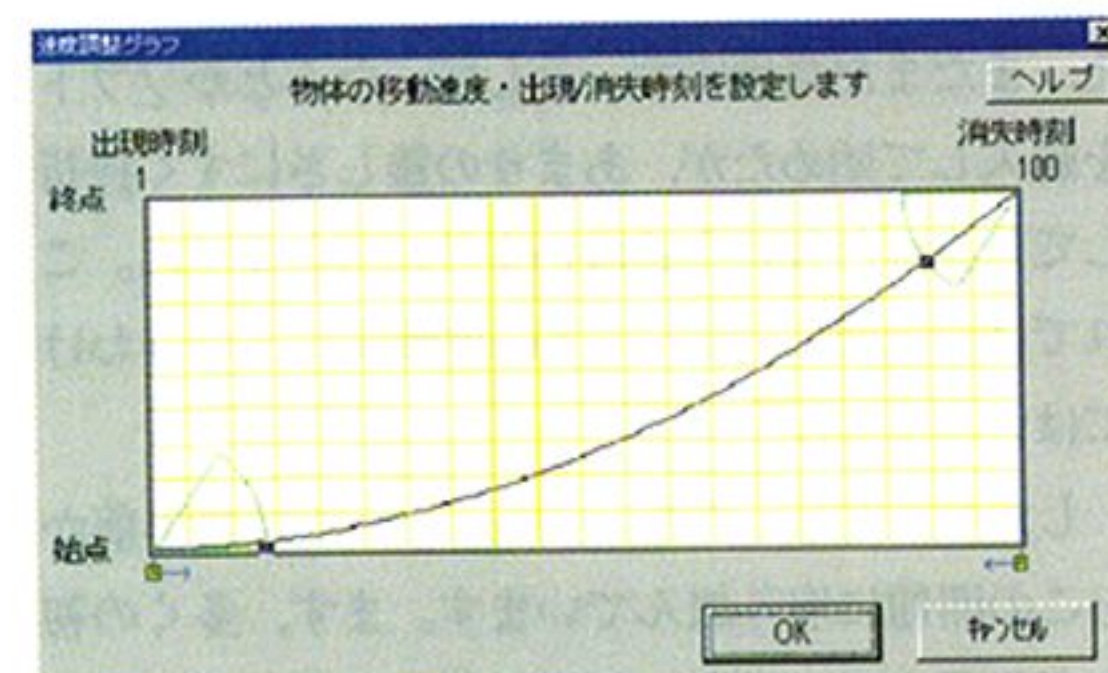
●パーツの選択

メニューの「パーツ」から「パーツ追加」を選ぶとパーツのカタログが表示されます（画面3）。この画面には一度に16種類のパーツしか表示されませんが，画面左の「種類」（現在「船体1」になっているところ）を変更すれば，ほかのパーツが表示されます。

この中から，好きなパーツを選択します。

●パーツの配置

パーツが選択されると，パーツが長方形に囲まれて黄色く表示されます（画面4）。これをBOX表示と呼んでいます。

この状態で，BOXの中心の赤いマーカーをつかんで動かすと，パーツの位置が変わります。BOXの各辺の水色マーカーを動かすと，拡大縮

画面6

画面8

画面10

画面7

画面9

画面11

小します。BOXの頂点の緑マーカーを動かすと，回転します（画面5）。

●その他

操作説明は，たったこれだけで終わりです。パーツの「選択」と「配置」を繰り返せば，あれよあれよという間になにかできてくるでしょう。一度配置したパーツを変更したり，左右対称に作るといった機能もいろいろありますが，それらは絶対に覚えないといけないものではありません。

とりあえず，物体が完成したら（画面6），保存してから終了します。

L1のモーションエディタの使い方

●モーションエディタとは

複数の物体を配置し，動きを決めたり，カメラや光源を設定するものです。パーツアセンブラより，一段階難しくなっていますが，画面の見方や操作感覚はほとんど同じなので，とまどうことは少ないでしょう。

●物体の配置

モーションエディタを起動したら，とりあえず動かない物体を配置しましょう。メニューの「物体」の「静止物体追加」で，パーツアセンブラで作成した物体を選択します。操作はパーツアセンブラと同じBOX表示で，好きな位置，向き，大きさに並べていきます（画面7）。

●動く物体の設定

「移動物体追加」を行うと，画面8のようになります。この状態で，左の赤い三角（始点）から右の赤い三角（終点）に向かってまっすぐ移動することになります。そこで，画面9のように始点，終点を動かしたり，紫色のマーカーを調節する（画面10）ことで，軌跡を自由に変更します。

●速度の設定

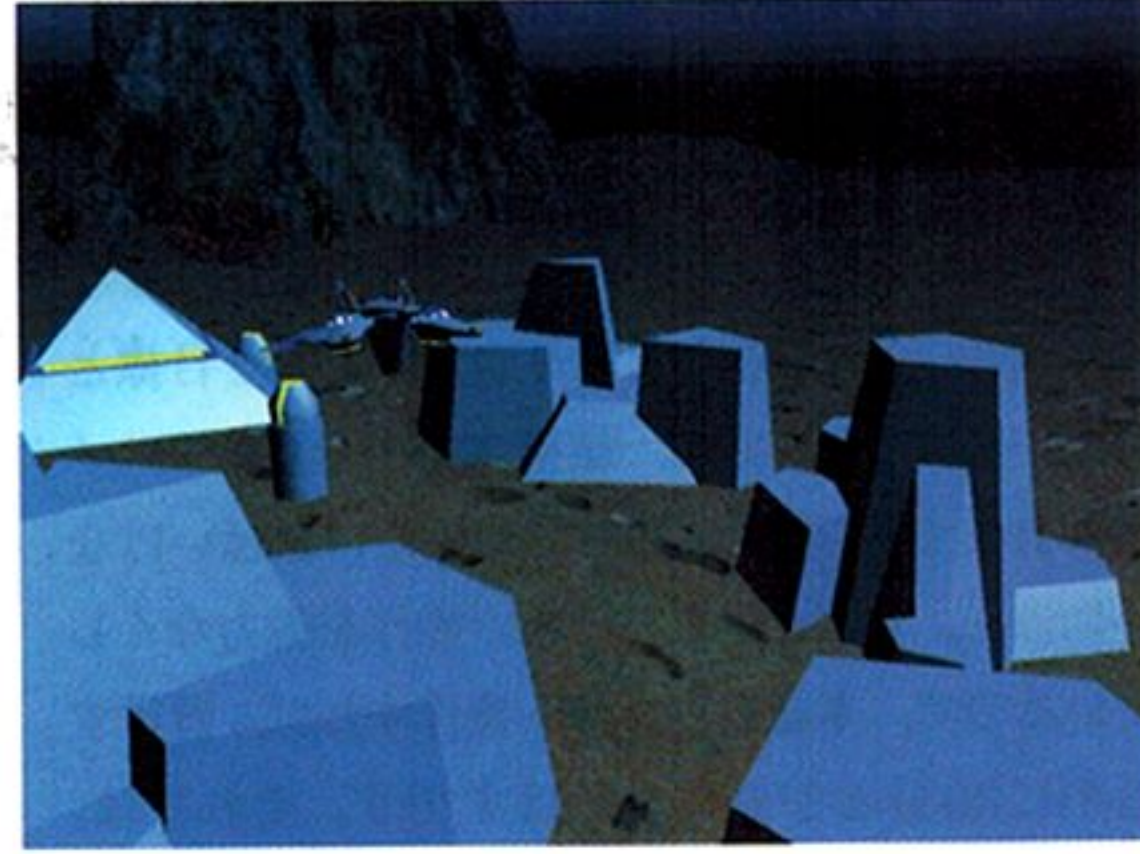
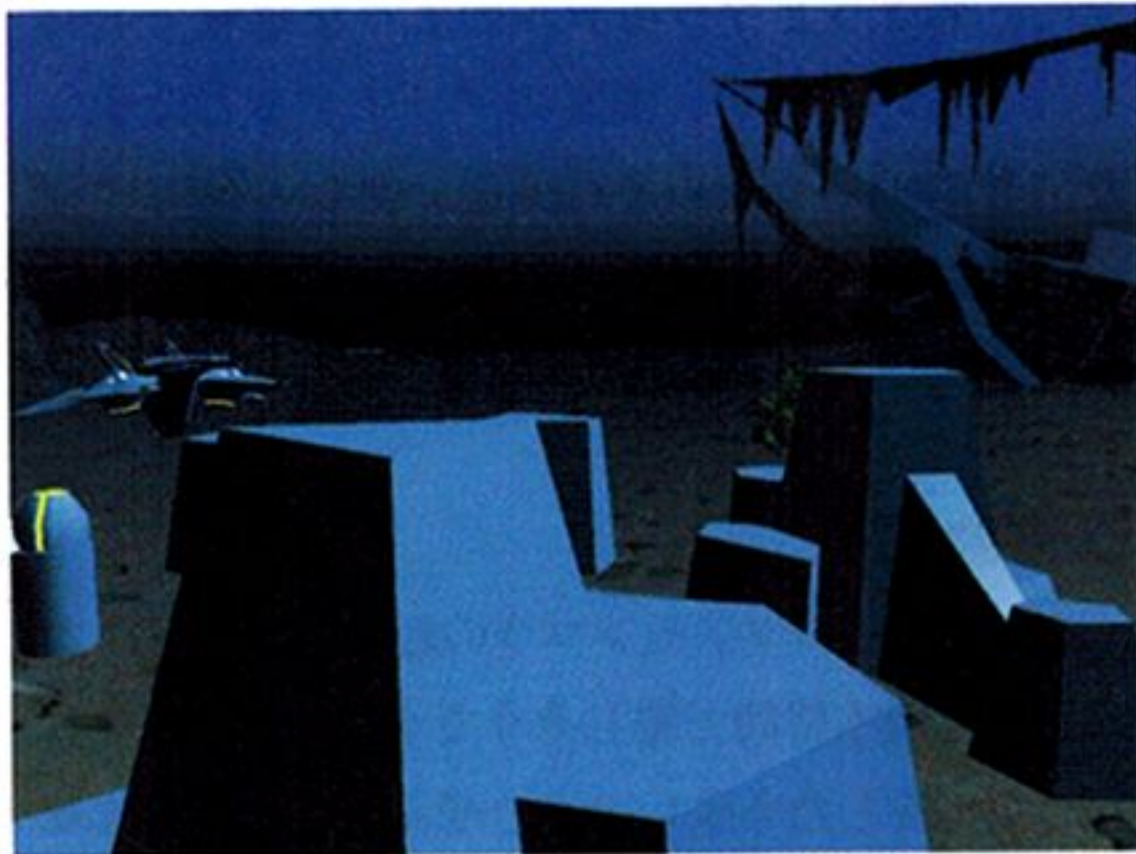
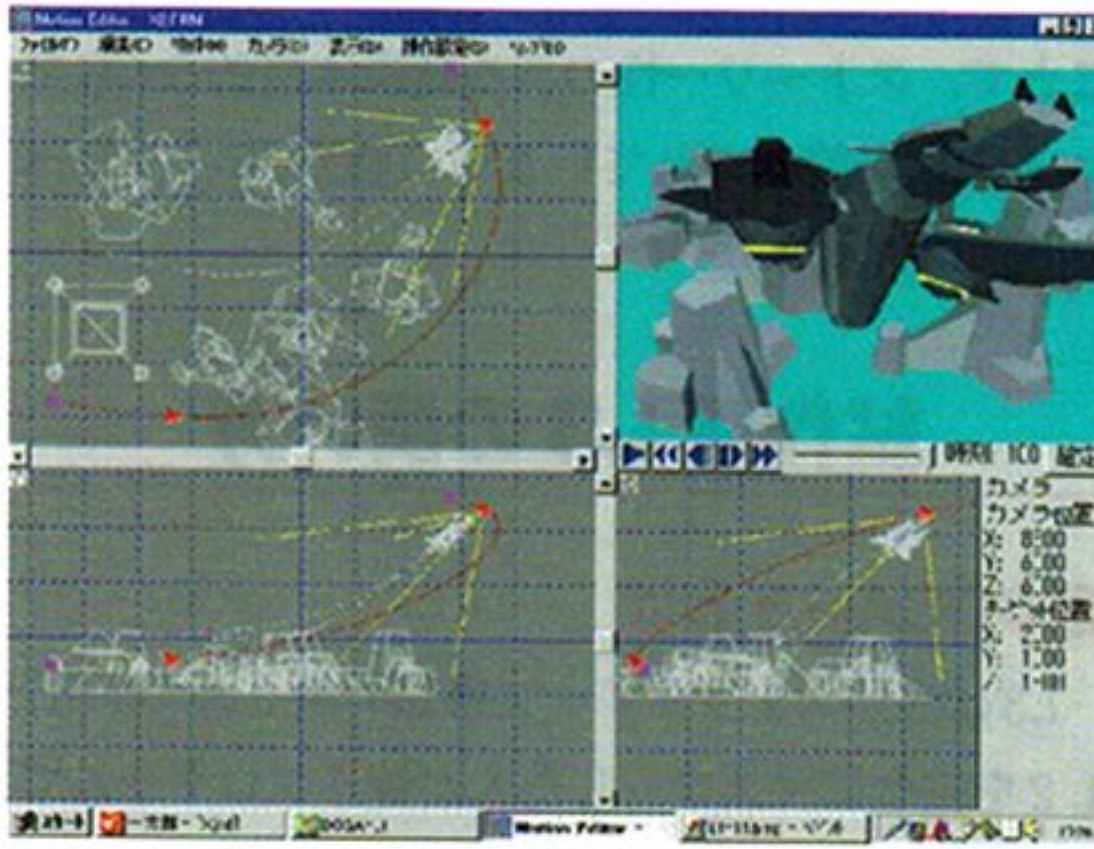
必要に応じて「物体」の「速度調整グラフ」（画面11）を出し，加速，減速を与えることもできます。このグラフでは，右上がりの傾斜がきついところほど速度が速いことを意味します。さらに，出現範囲を調節し，途中から現れたり，途中で消滅するような物体も設定できます。

●その他

「カメラ」の「カメラ移動設定」を実行することで，移動物体と同様にカメラの始点と終点が表示され，カメラを動かすこともできます（画面12）。その他，光源の向きの設定，カット全体の長さの調整などの機能もありますので，少しずつ覚えていってください。

L1のアニメーション表示

●アニメーション表示について

物体の形状デザイン，動きのデザインができたら，ちゃんと作画して，アニメーションを表示させましょう。ここでは，作画するモーションの順

画面12

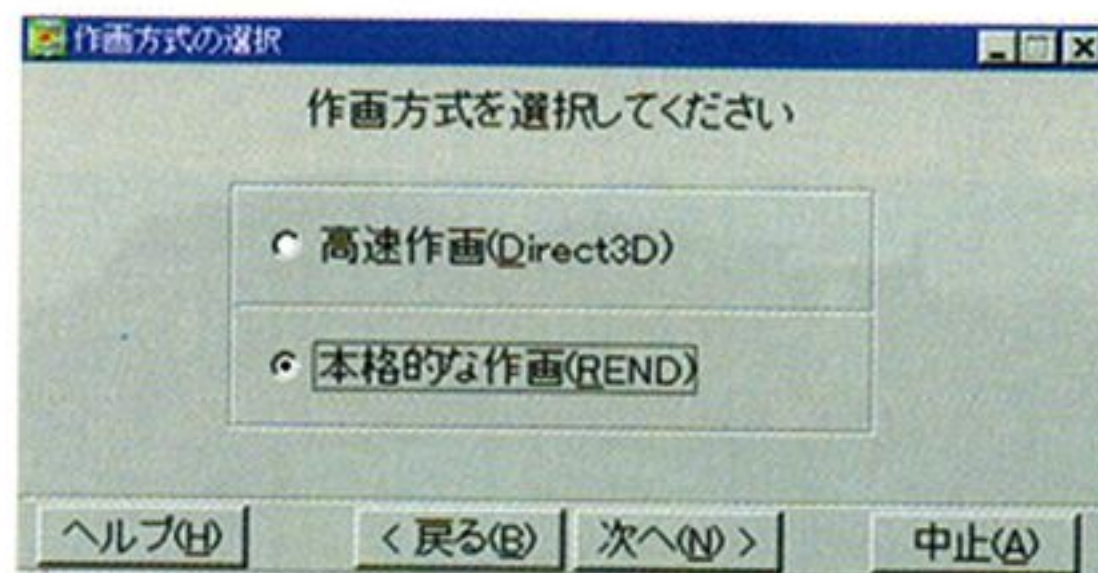
画面13

番を指定したり，背景を選択したりしますが，画面の指示に従って設定するだけなので，いたって簡単です。作画が始まると，かなりの時間待たされますが，全部自動的に行われるので，ほっといても構いません。

●作画させるカットの選択

メインメニューから「アニメーションの表示」を選択すると，モーションの選択画面（画面13）になります。モーションエディタで作成したモーションデータがカタログ表示されますので，作画したいデータを選択します。このとき，複数のモーションを選択すると，その順番でそれらのカットがつながった長いアニメーションを作ることもできます。

●色や背景の設定

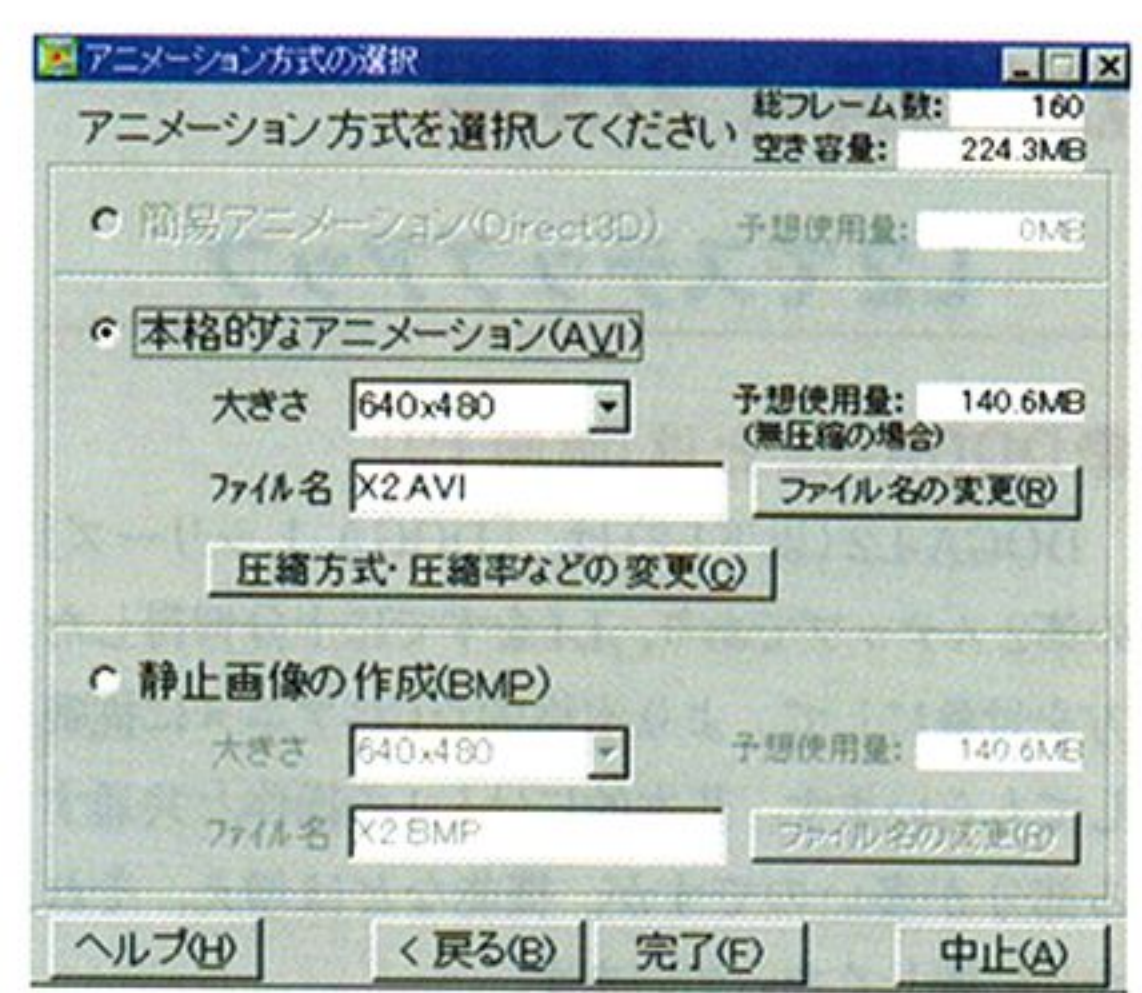
指定されたモーションの中で使われている物体の一覧が表示されますので，それぞれの物体の色を決めてください（画面14）。今回は，海の中のイメージですべて青にしてみました。

また，背景も画面15のように，いろいろ用意されています。

画面14

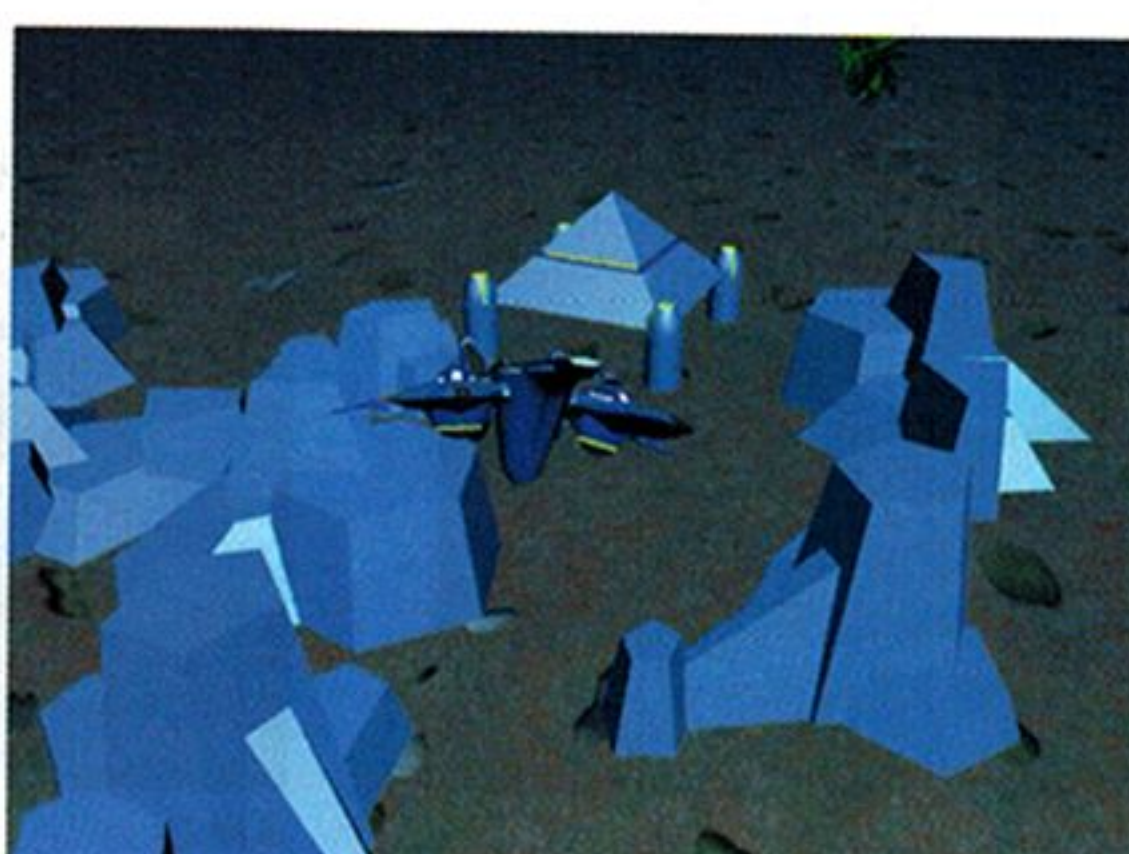
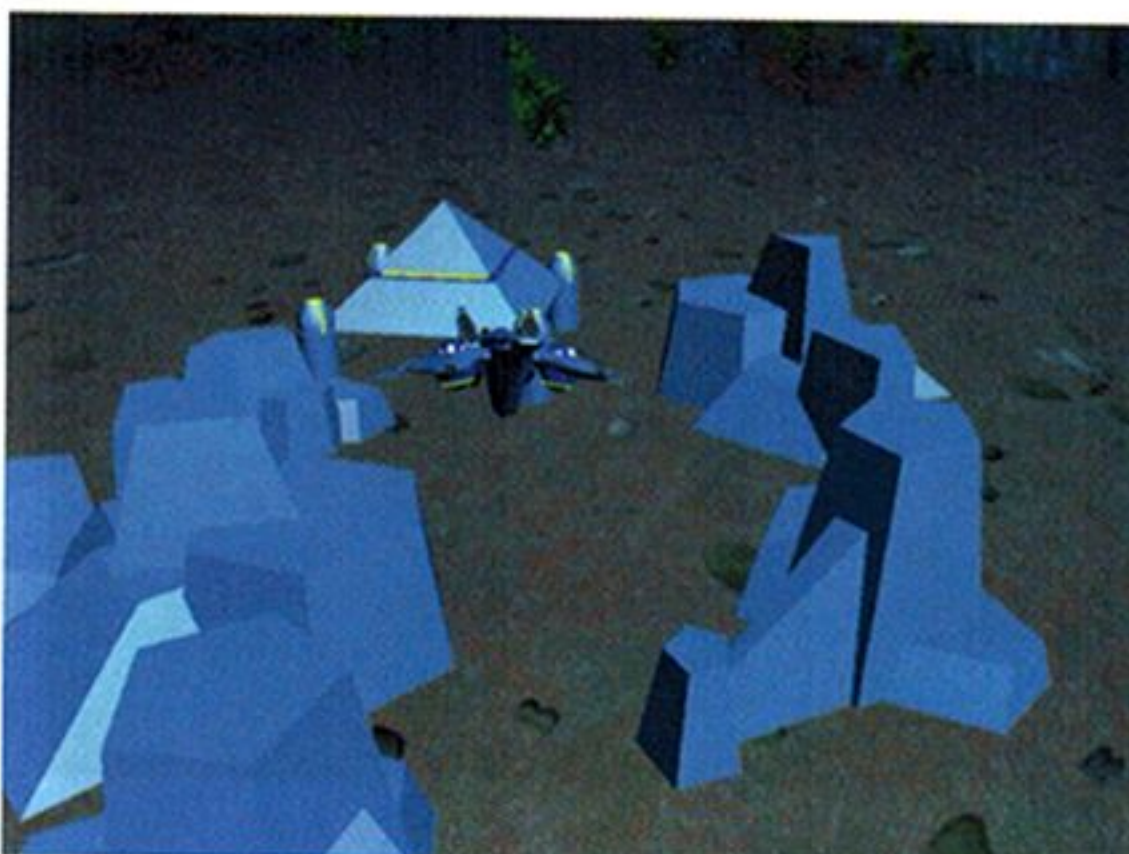
画面15

画面16

●アニメーションの作画開始

引き続いて，作画方式（画面16），アニメーション方式（画面17）を選択すると，いよいよ作画を始めます（画面18）。

あとは自動的に処理され，作画がすべて終わると，アニメーションが表示されます。これで，CGアニメがひとつ作れたわけですが，いかがです？　簡単でしょう？

画面17

画面18

画面19

DOGA-L2/3編

L2でステップアップ

●DOGA-L2とは(画面19)

DOGA-L2(以下L2)は,「DOGA-Lシリーズ」の第2ステップであり,L1をすでに十分習得した方を対象にして,より本格的なCGアニメに挑戦してもらいます。基本的にはL1の操作と共通する部分が多いのですが,機能などは増え,それなりに難しくなっています。

今回初めてLシリーズを知った読者は,もちろんL1のほうから習得していただくことになるので,L2を使いだすのは,少し先になるでしょう。ですから,今回は詳しい使い方は省略し,L2でどのようなことができるようになるのかを紹介しましょう。

●多彩な材質表現

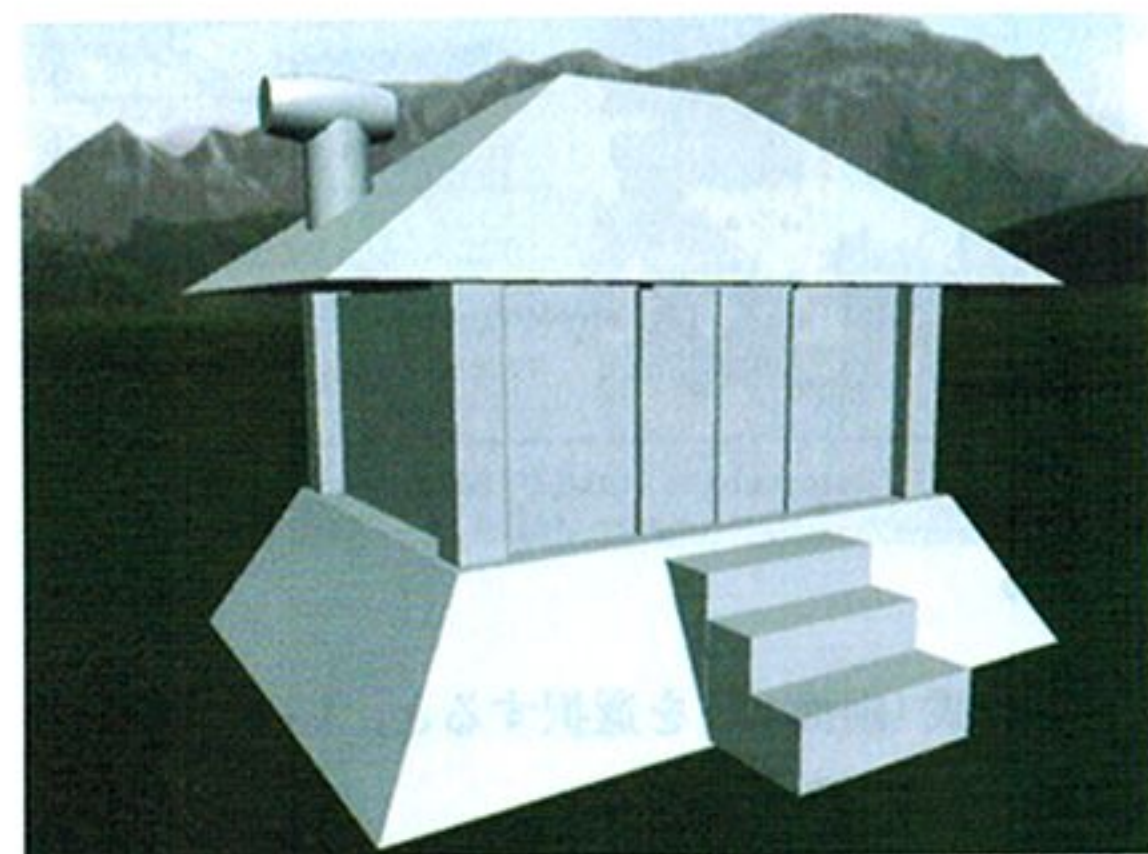
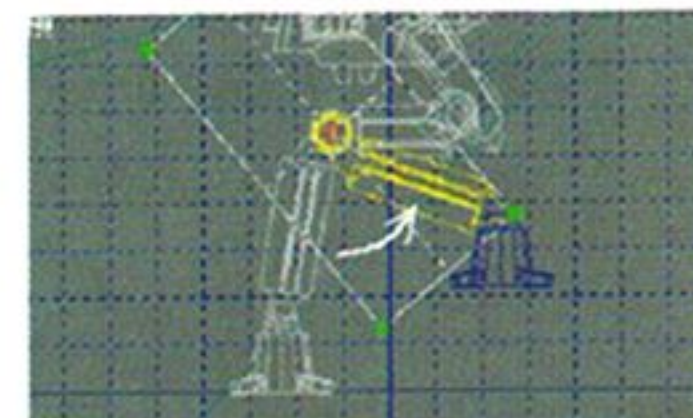
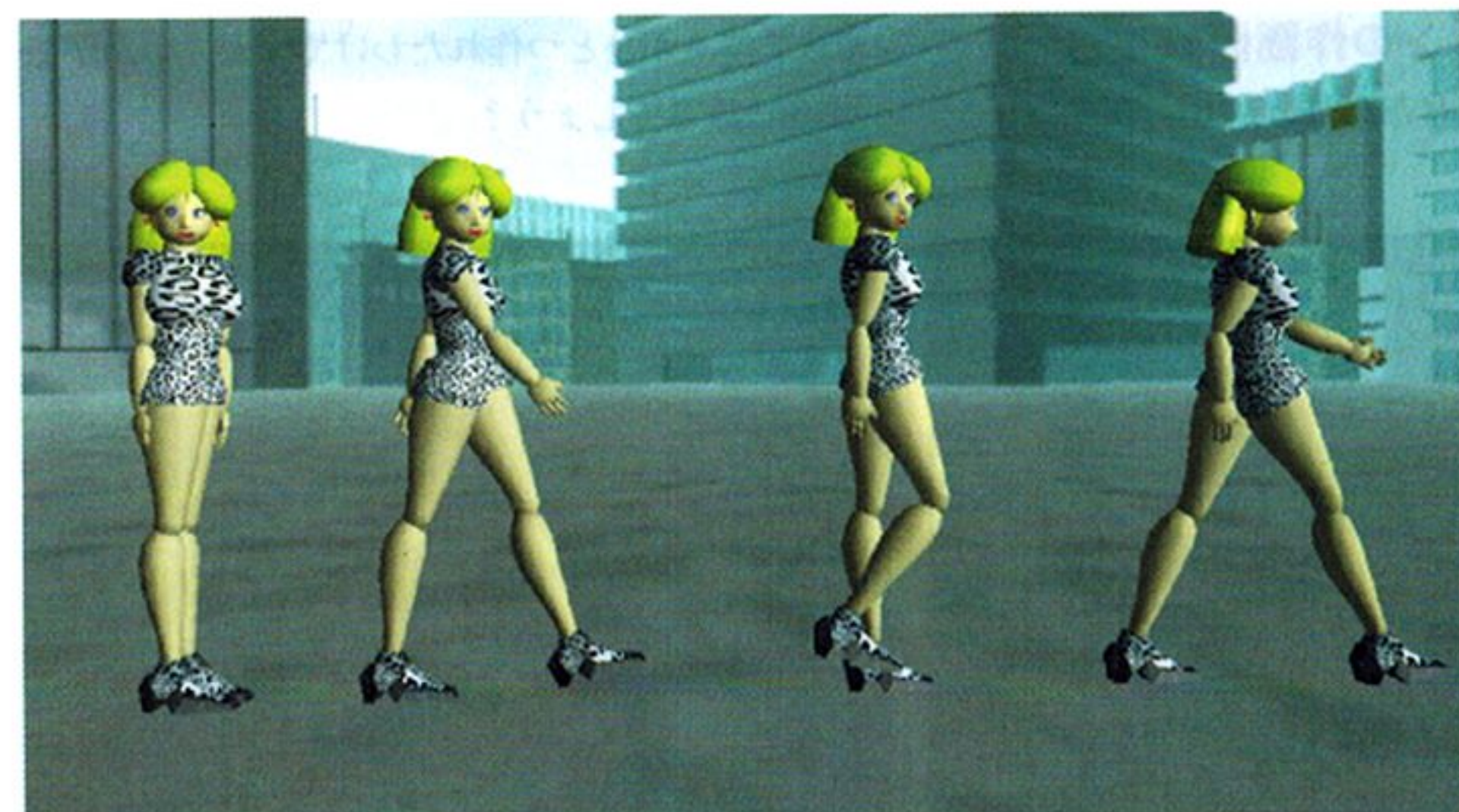
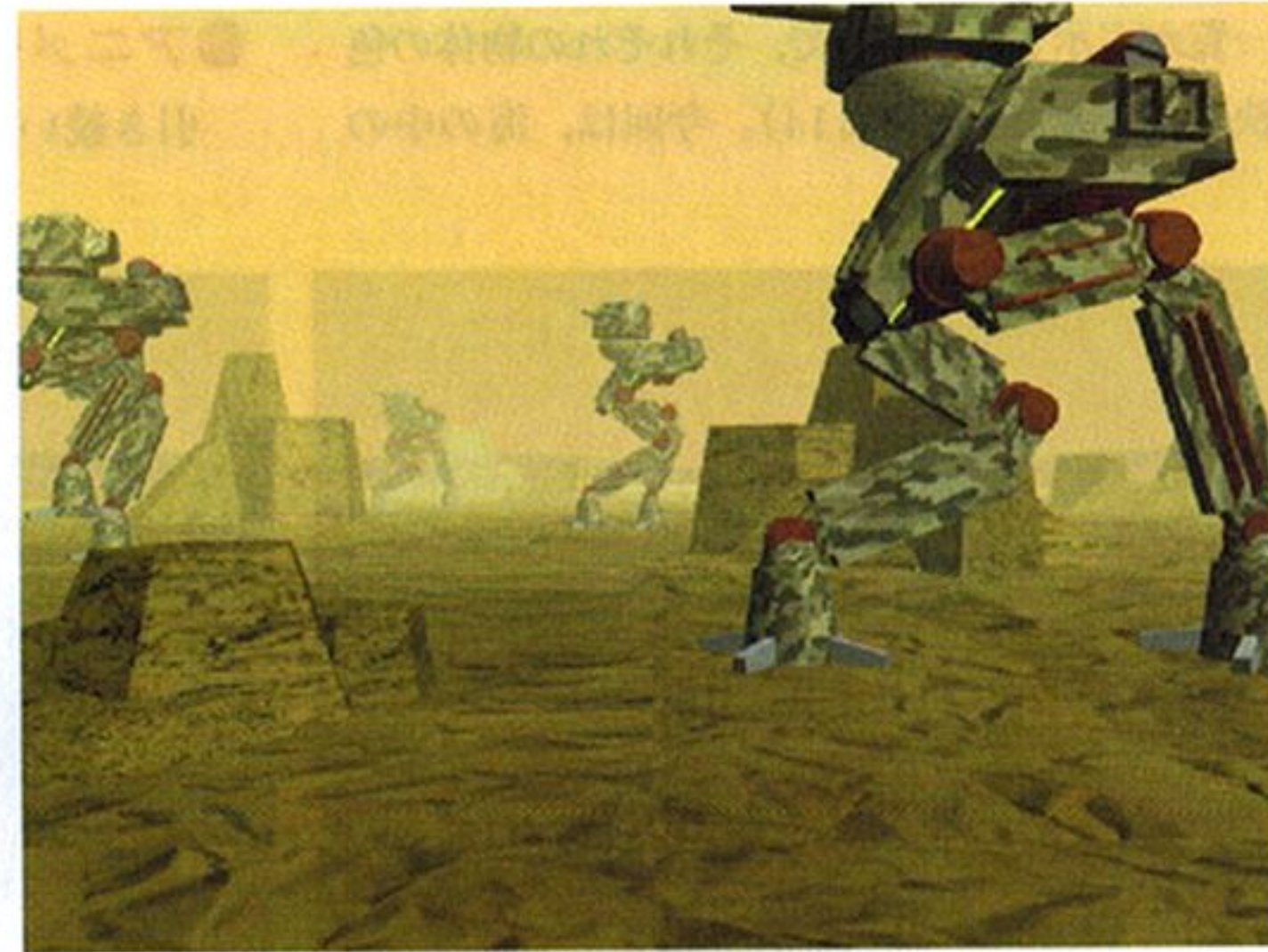
L2では,パーツアセンブラにおいて,パーツを選ぶ際,そのパーツの色や模様,質感などを設定することができます(画面20,21)。

これだけで,CGアニメとしての表現力は断然アップします(画面22)。

●多関節物体

多関節物体とは,ロボットの腕や足のような関節を持った物体です(画面23)。多関節物体を扱うことで,ただ位置が変わるのではなく,ポーズを変えながら歩くといった表現が可能になります(画面24)。

●その他

その他L2では,有色光源や霧(霞,砂煙)(画面25)といった表現や,ズーミングなど,少し凝ったカメラワークも可能です。これらを駆使することで,ストーリー性のある結構本格的なCGアニメ作品が制作されています(画面26:「リベンジ」山本 経孝)。

試作版L3の使い方

●DOGA-L3とは

L3は当然ステップアップ構想の第3弾であり,L2を完全にマスターした人が使うプログラムです。L3では,L2の個々のツールがより複雑になるだけでなく,新しくAVエディタ(仮称)が加わります。

AVエディタは,複数の動画を編集し,BGMや効果音を入れ,テロップなどを加えて,映像作品として仕上げるツールです。AVエディタのメイン画面で,動画の編集と音をづけを行い,それ以外の機能は,別途呼び出すようなシステムになっています。別途呼び出される機能としては,効果音加工ツール,効果音データベース,BGMデータベース,テロップ,スクロール,エフェクトなどが予定されています(あくまで予定)。

●試作版の注意

とはいっても,L3はまだまだ開発途中であり,今回CD-ROMに収録したのも,AVエディタの試作モデルにすぎません。バグが多いといったレベルではなく,まだできていない機能があるというか,まだ仕様すら固まっていない状態です(外部機能はまったくできていない)。今回の試作モデルをたたき台にして,L3のAVエディタをどんなツールにするのか検討したいと思っています。

今回,そんな試作モデルを公開したのは,これ

画面19

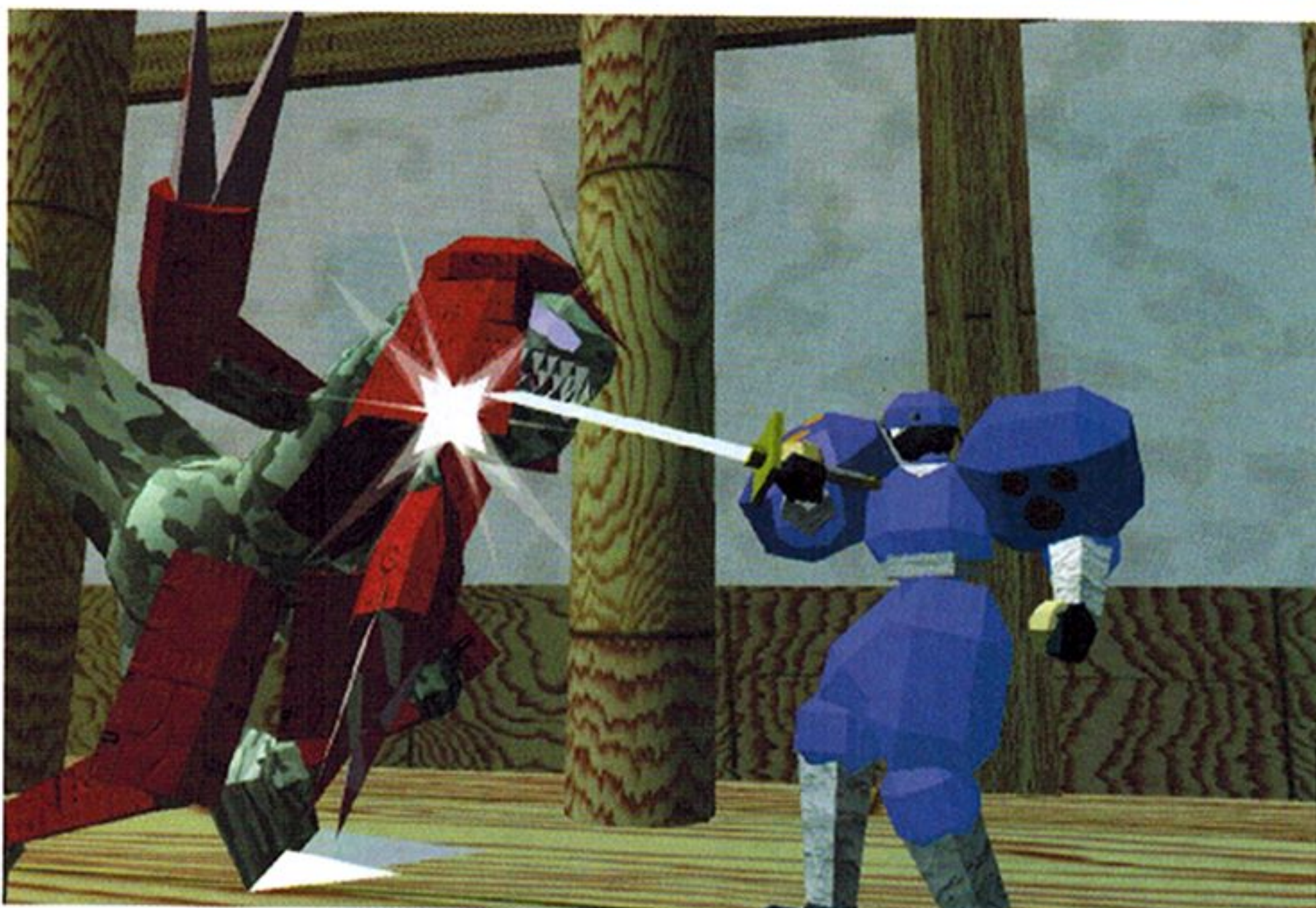
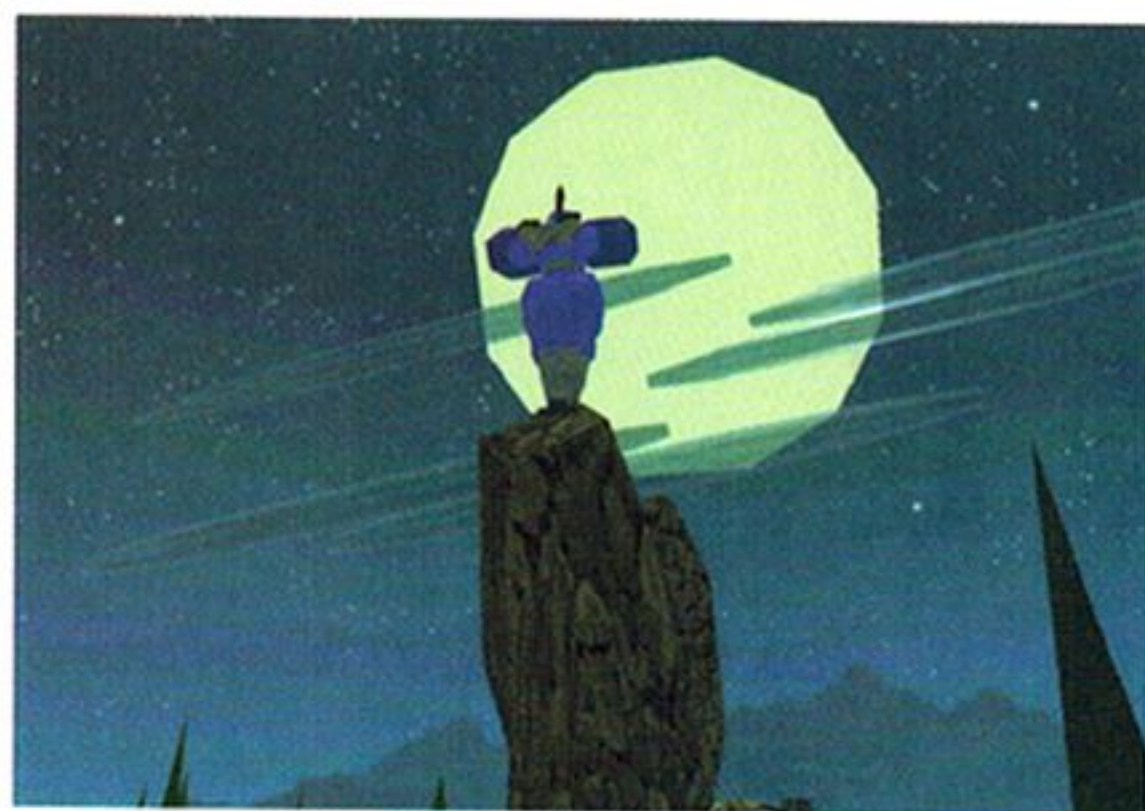
画面20

画面21

画面22

画面23

画面24

画面25

画面26

を使ってくださいという意味ではなく，"こんなものを作ろうと考えているが，なにかよいアイデアはないかい？"と広くご意見を募集するためです。ですから，L3の開発に，参加，協力の意志のある方だけ，触ってみてください。それ以外の方は，残念ながら，まだ全然実用性がないので，インストールするだけ無駄だと思います。

とにかく，まだろくに動かない，操作性が悪い，ものすごく遅い，機能が足りないといったことは，あらかじめ覚悟してください。

●操作概要

起動の前に，とりあえずAVIファイルが複数ないと編集の意味がありませんので，L2で数カット作っておきましょう（ひとつの長いアニメーションではなく，カットごとに作画しておく）。ついでに適当なWAVEファイルもいくつか用意しておきましょう。

まず起動ですが，まだメインメニューなどはできていないので，「aviedit.exe」を直接クリックしてください。AVIファイルを読み込むのは，「ファイル」の「クリップを追加」ですが，複数のカットを読み込むのにいちいちこんなことはしていられないので，キーボードの「INS」を使うことになるでしょう。

現在稼働する機能は，

・映像のカット＆ペースト
・WAVEファイルの貼り付け
・ワイプ/オーバーラップの作成

といったところです。ヘルプを参考にして，操作してみてください。

ひととおり編集が終われば，「ファイル」の「プロジェクトに名前を付けて保存」で編集データを保存し，「ムービー」の「ムービー作成」を実行し，全部を1本のAVIファイルに作り直します。この作業は，非常に時間がかかるので，暴走と勘違いしないでください。

最後に

このL3，いったいいつ完成するかといえば…

…いつなんでしょうねぇ。希望としては，年末に完成して来年2月頃発表したいのですが。なんせLシリーズは，市販ソフト並みに開発費がかかっているのに，無料で配るといった暴挙を行ってしまった結果，すでに開発費は底をついている状態です。宝くじでも当たらない限り，少なくとも無料配布はもうしないでしょう。

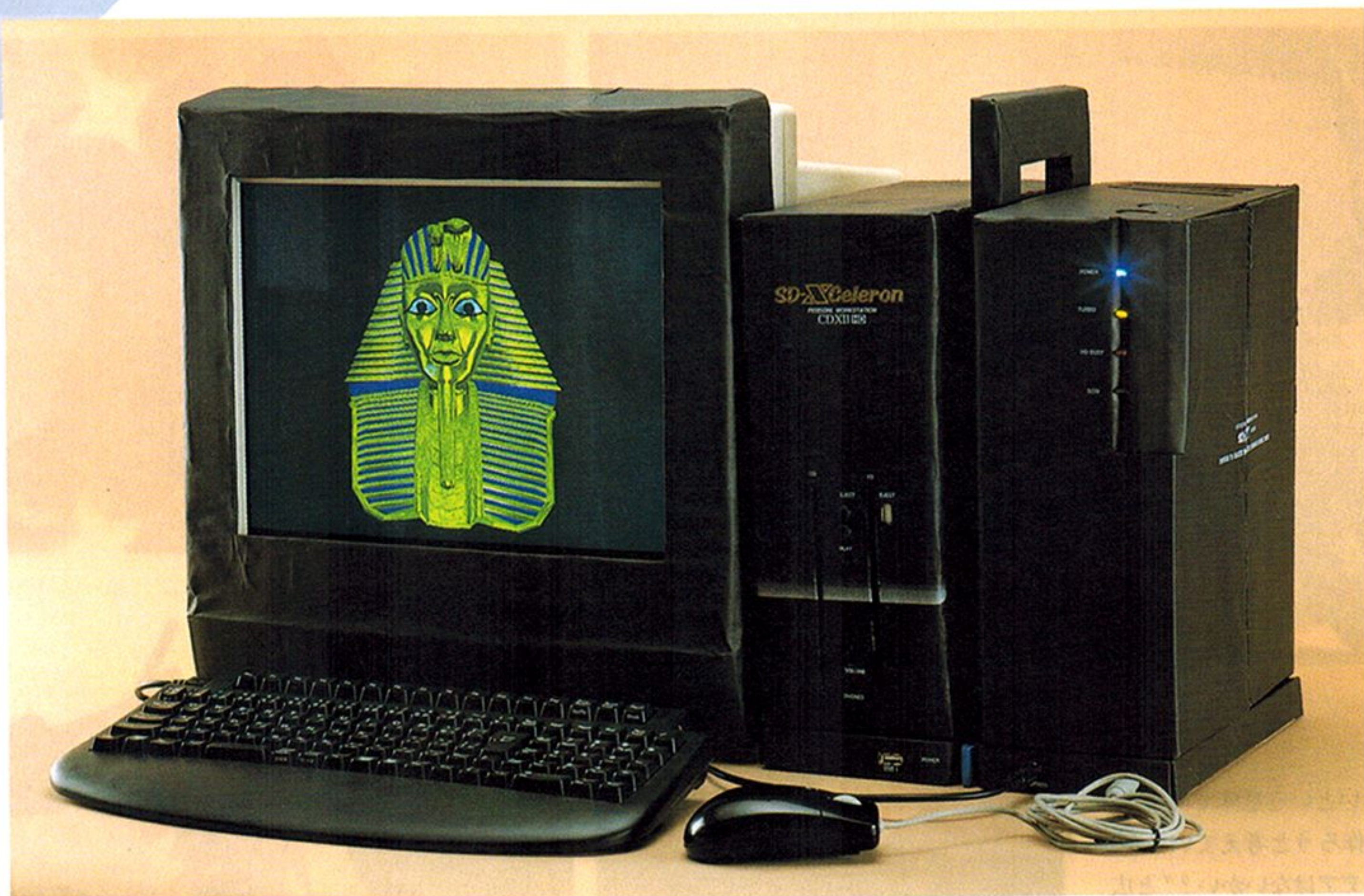
またL3には，BGMや効果音のデータ集をつけたいのですが，この辺は当方だけでは用意することができませんので，広く皆さんからデータを提供していただきたいと考えています。その他いろんな意味で，L3は皆さんのご協力なしには完成させることができません。今後ともよろしくお願いします。

http://www.doga.co.jp/ptdoga/

Hardware Geeks

PC/AT互換機用 オリジナルATX(?)筐体の製作

SD-X Celeron CDX II

Text：菊地 功 / Kikuchi Isawo

おねえさん：やあやあ，みんな元気だった？ ついにというか，やっとというか，Oh!Xが復刊したわねぇ。

ポン太君：ゴホゴホ！

おねえさん：そう，ポン太君も嬉しいの。しかも「Oh!X」の「X」は「未知」の「X」ってことで，プラットフォームはなんでもあり。

ポン太君：ゴホゴホ。

おねえさん：ふ～ん，ポン太君はWindowsなんだ。まあここはDOS/V magazine編集部内だし，編集の(U)氏もどっちかっていうとWindows派だしね。

ポン太君：ゴホゴホゴーホゴホ！

おねえさん：え？ でもマシンがカッコ悪い？ 確かにね，あのマシンに比べたら，カッコイイマシンなんてめったにないわよね。とりわけPC/AT互換機の筐体ときたら…。おかげでちょっと「変な形」してるだけのiMacがバカ売れしてるみたいだし……。

ニョッポさん：……。

おねえさん：あらニョッポさん，こんにちは。

ニョッポさん：………！

おねえさん：ないなら作れ。それがXの精神だ！ ってねぇ，そんなこといっても，簡単にはできないわよ。フロントマスクの金型だけでいくらすると思ってるの？

ニョッポさん：………。

おねえさん：これがあれば大丈夫，って，それダンボール箱じゃないの。そんなのでいったい……。

ニョッポさん：………，………。

おねえさん：ふんふん，(U)氏のサーバがダンボール箱の中に放り込んであって，それが2個積み重ねられているのを見てピピンときた？ だって，ポン太君は「カッコイイ筐体」がほしいっていってるのよ？

ポン太君：ゴホゴホ！

ニョッポさん：……！

おねえさん：まあ，そこまでいうなら……。でもさ，中身はどうするの？ 箱だけ作ってもしようがないでしょ？

ポン太君：ゴホゴホゴーホ！

おねえさん：え？ ちょうどこのあいだ買ったパーツがある？ まあ，ポン太君ってバラ組みするんだ。なになに？ P2BマザーとCeleron 266，Millennium G200？ なかなかマニアックなのねぇ，ポン太君。

ポン太君：ゴッホン！

おねえさん：しかもCeleronは103×4=412MHz駆動？ やあねえポン太君。400MHzオーバーなんて当たり前よ。Celeronを規定クロックで動かす人がいるわけないじゃないの(偏見)。電圧上げてでも448MHzで動かさなきゃ！

ポン太君：ゴホ！(ガ～ン)

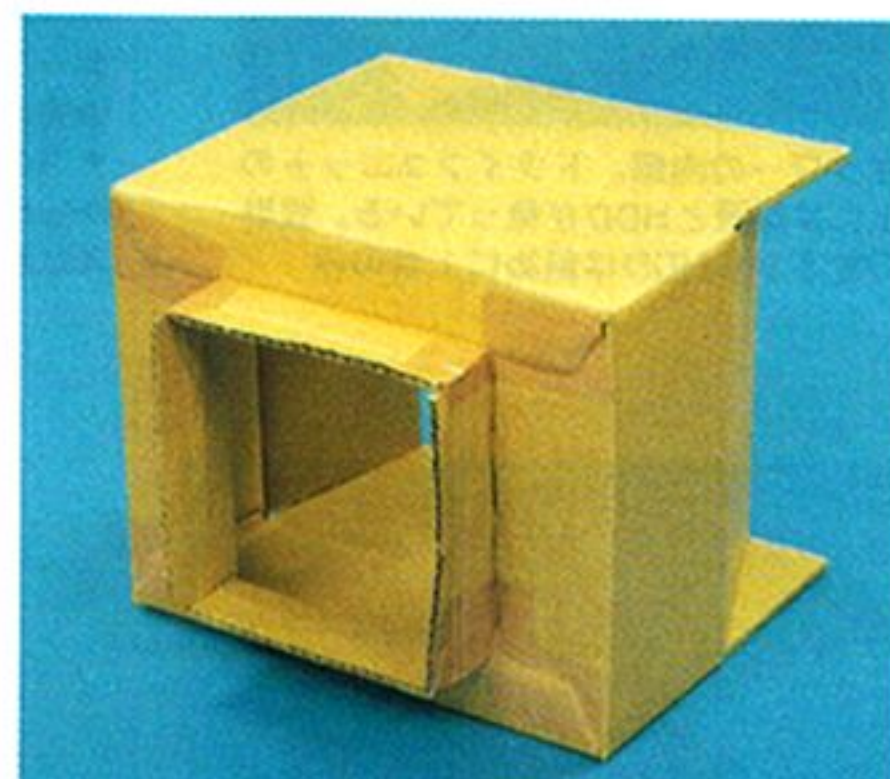
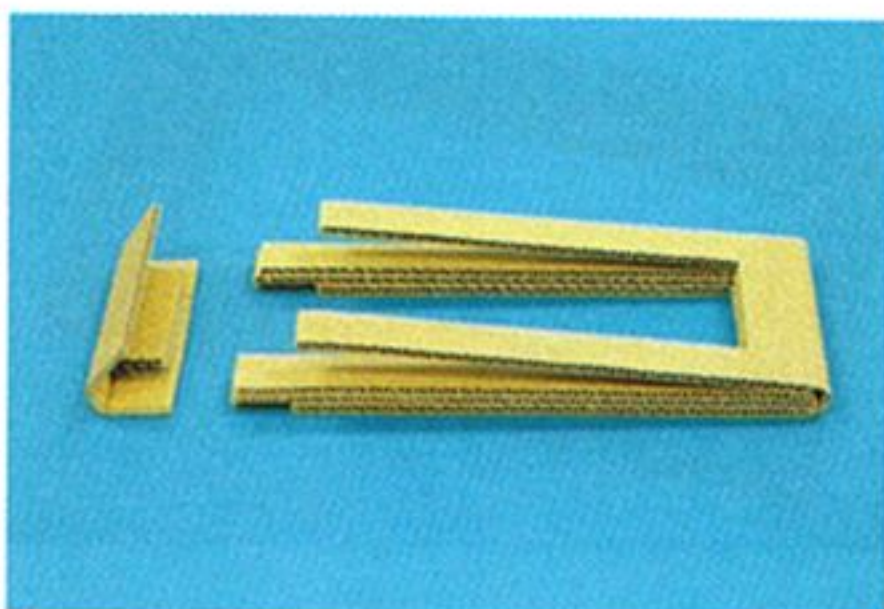
おねえさん：でも，本当に大丈夫なの？ ニョッポさん。

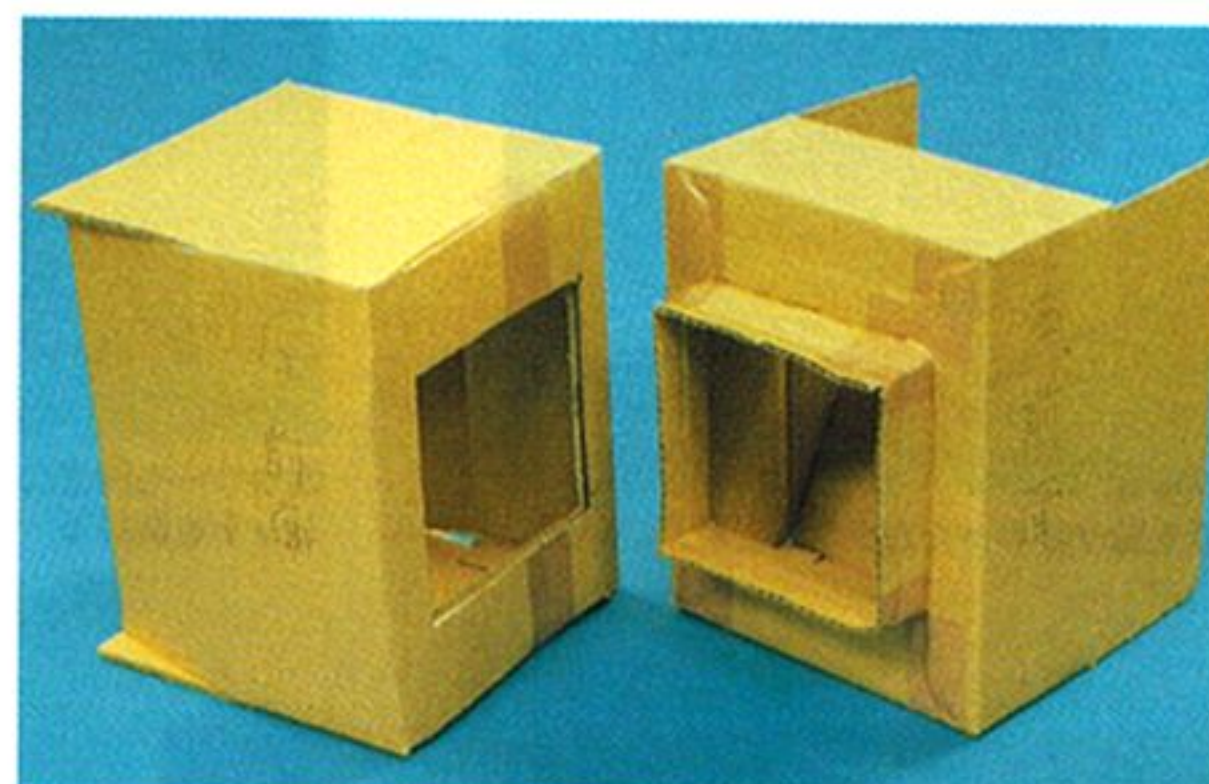
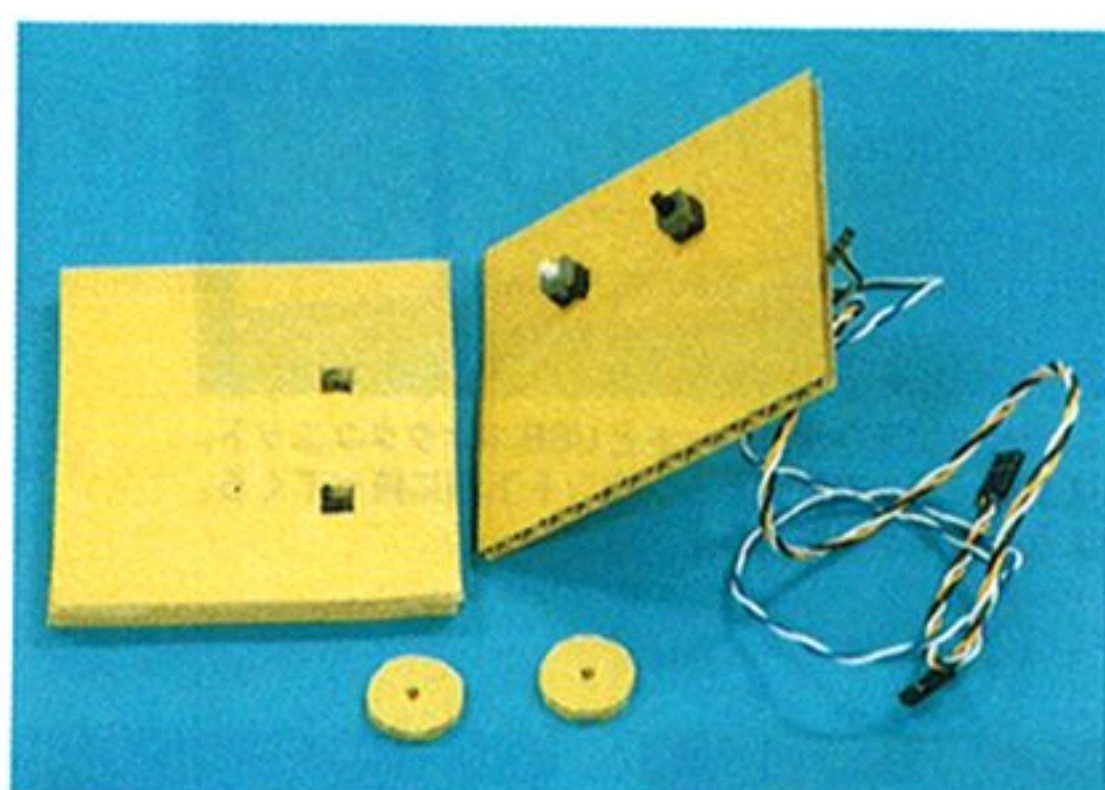
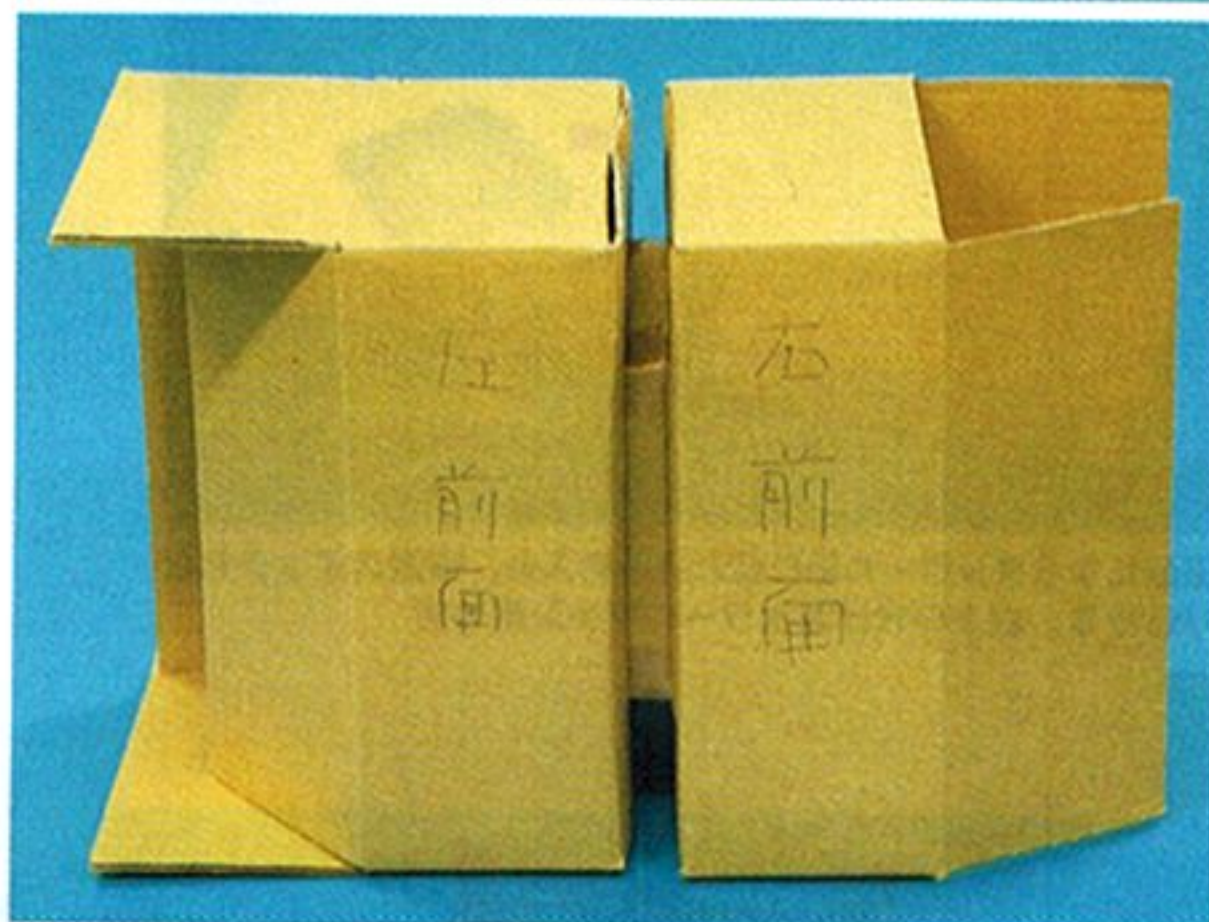
ニョッポさん：……！

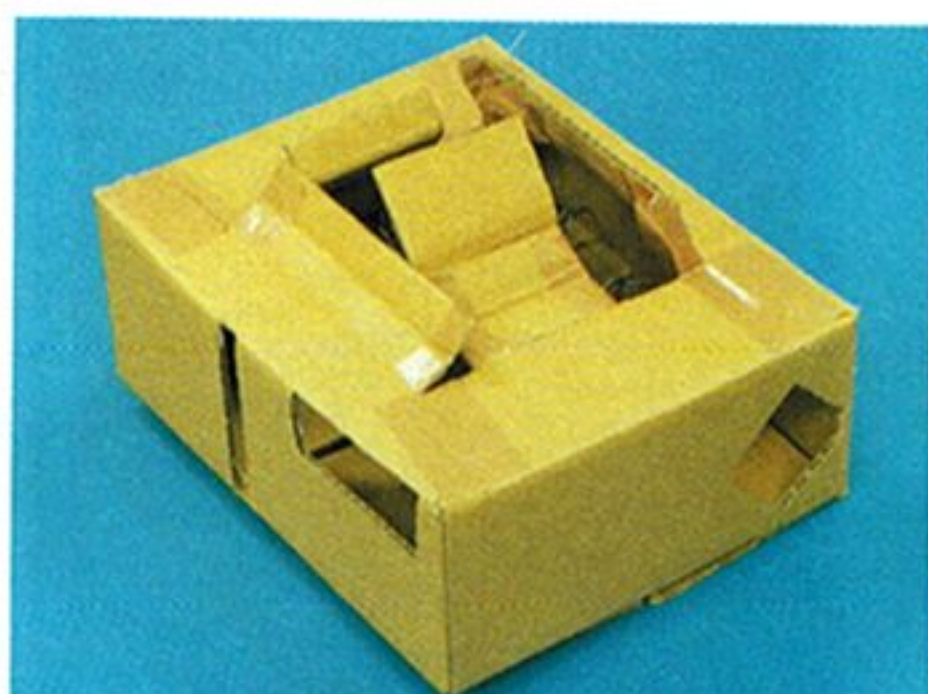
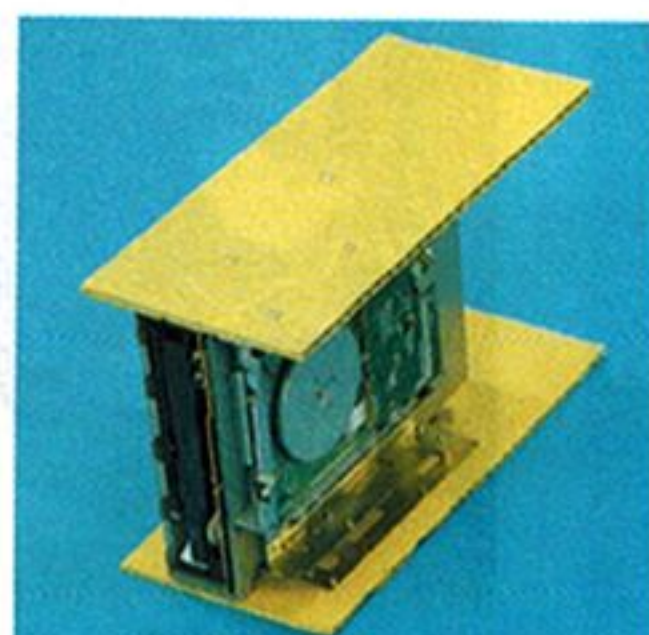
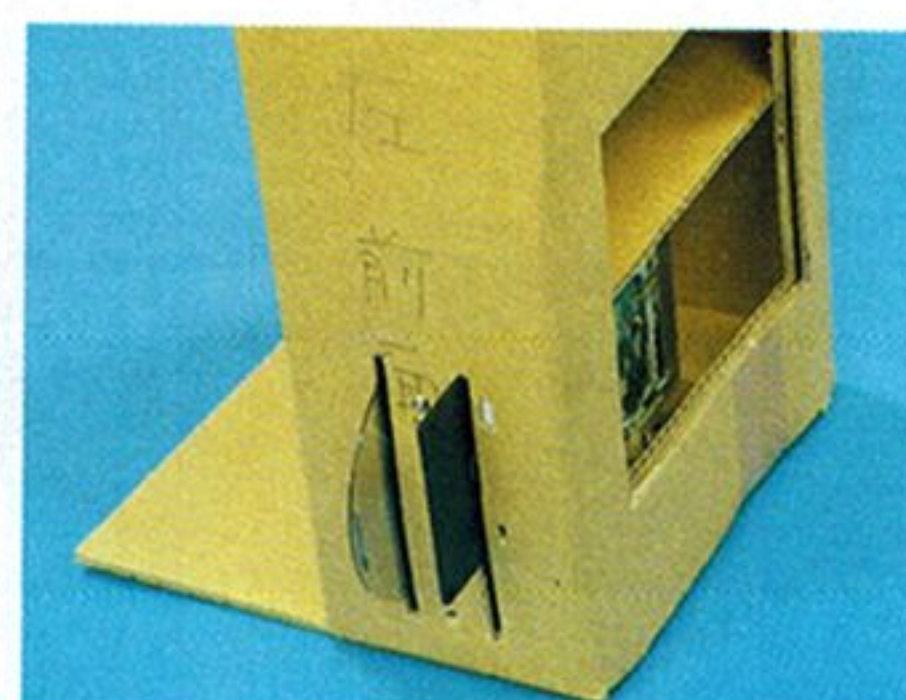
おねえさん：じゃ，ま，適当にがんばってね。期待してないから。

デッキルッカナ，デッキナイッカモ，ハテサテホホ～♪

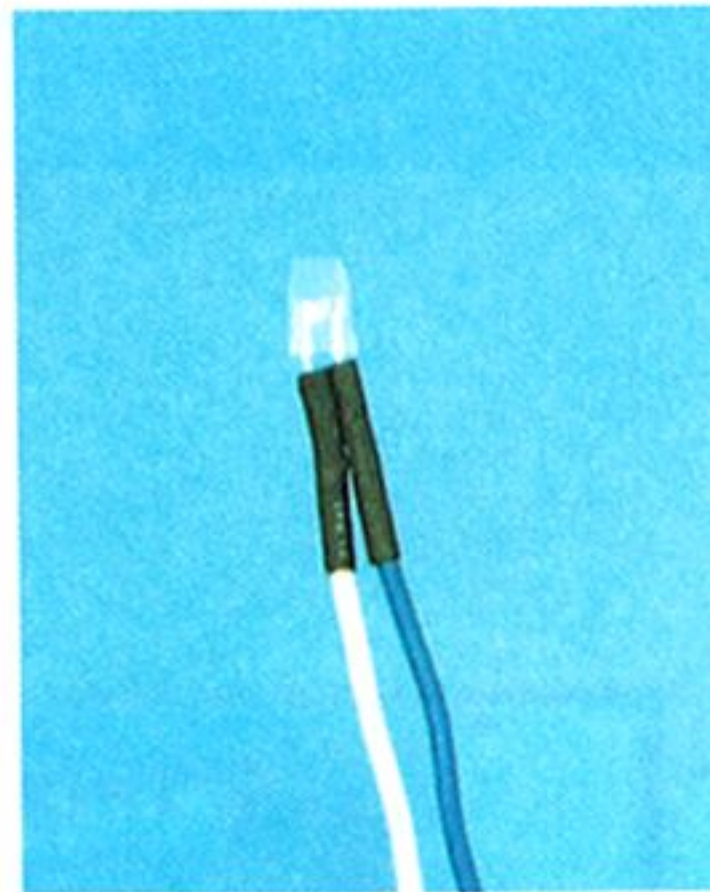
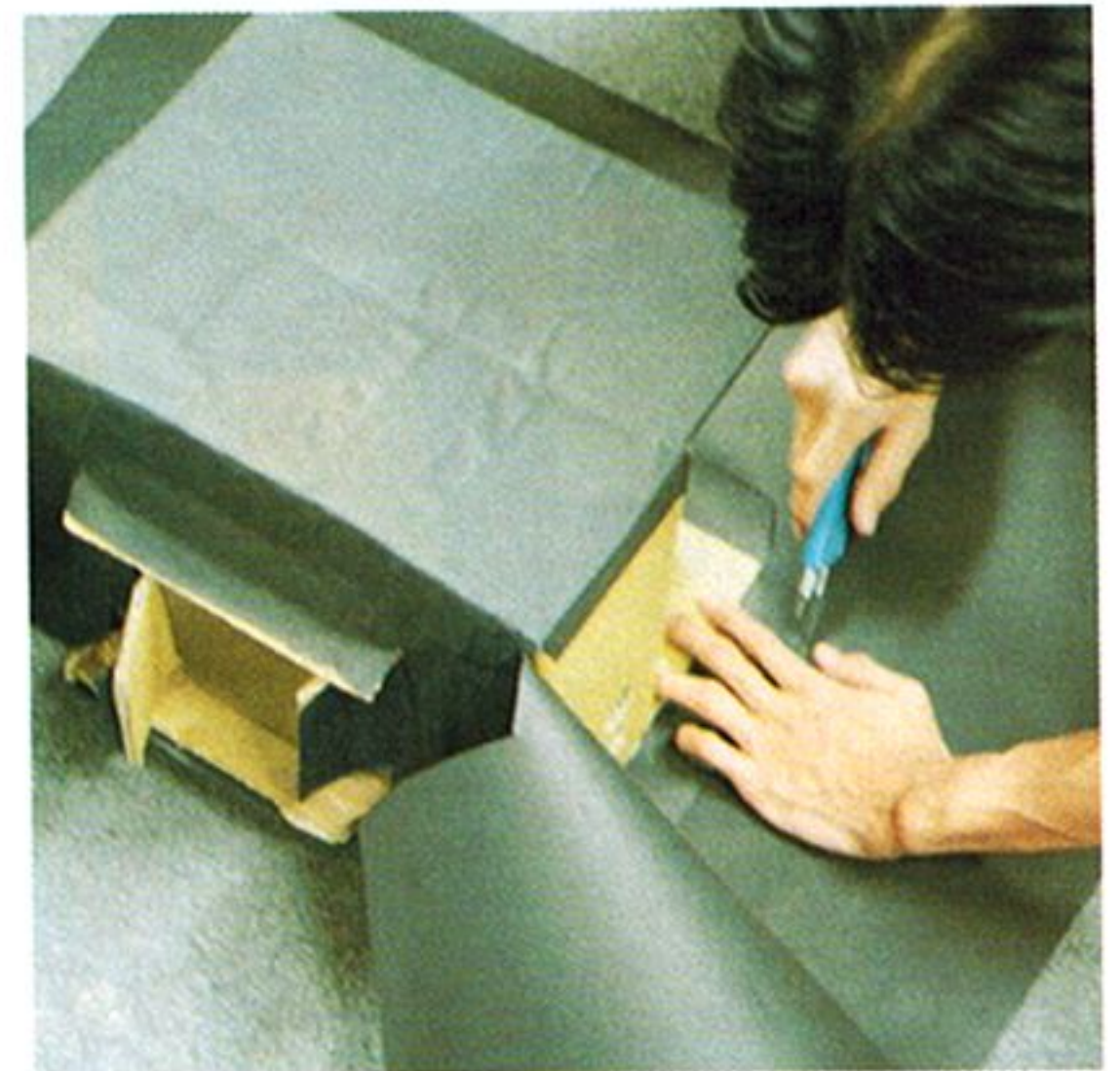
これが全パーツ。まさにこれから「自作するぞっ」って感じだ

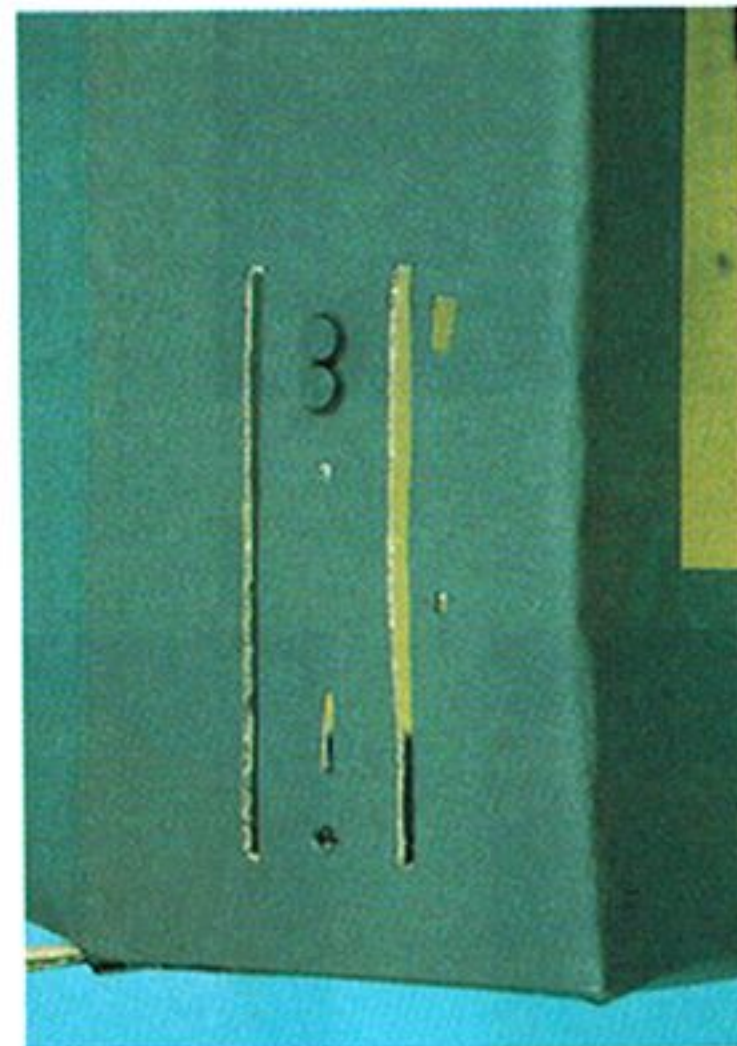
タワー部となるダンボール箱。高さ(タワーの幅)が少々大きすぎるので，マザーにカードを差した高さにあわせて少し詰める(左)

左右のタワーを連結するブリッジ部を作る。写真は右側のタワー

キャリングハンドルもダンボールで作ってしまおうという，ちょっと大胆な作戦

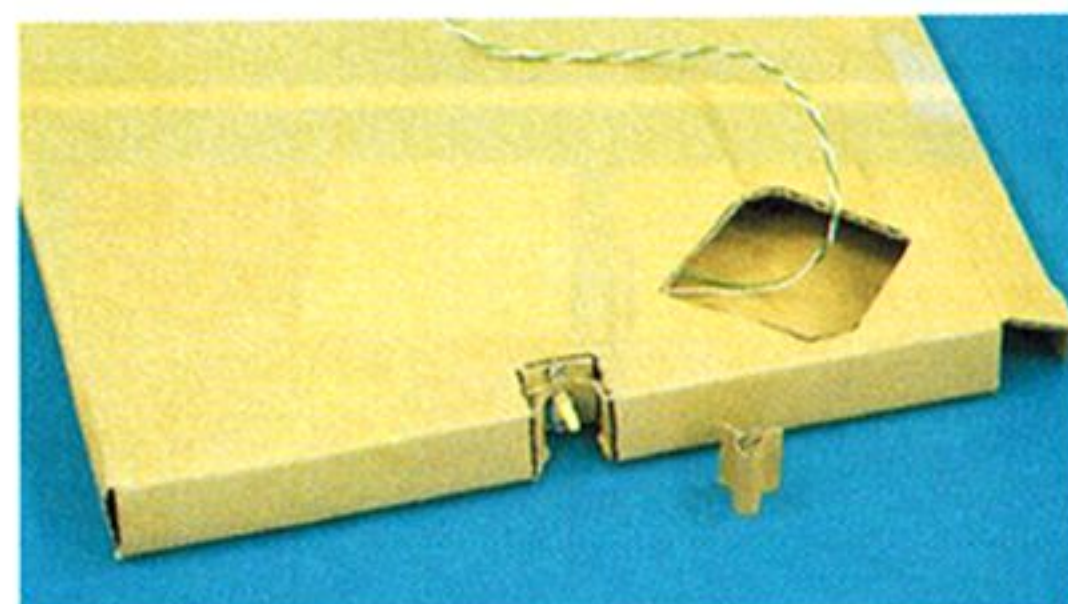
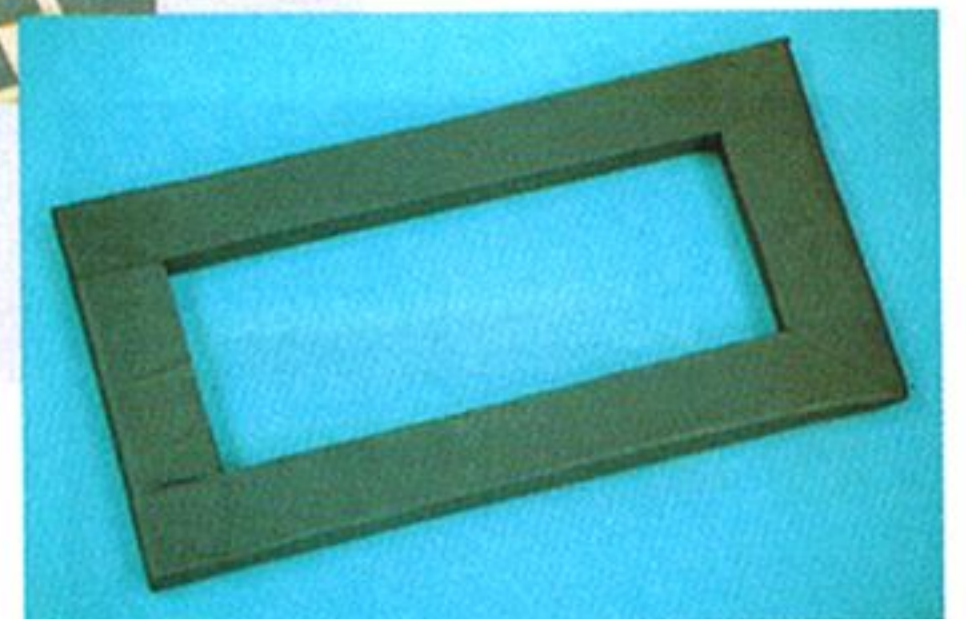
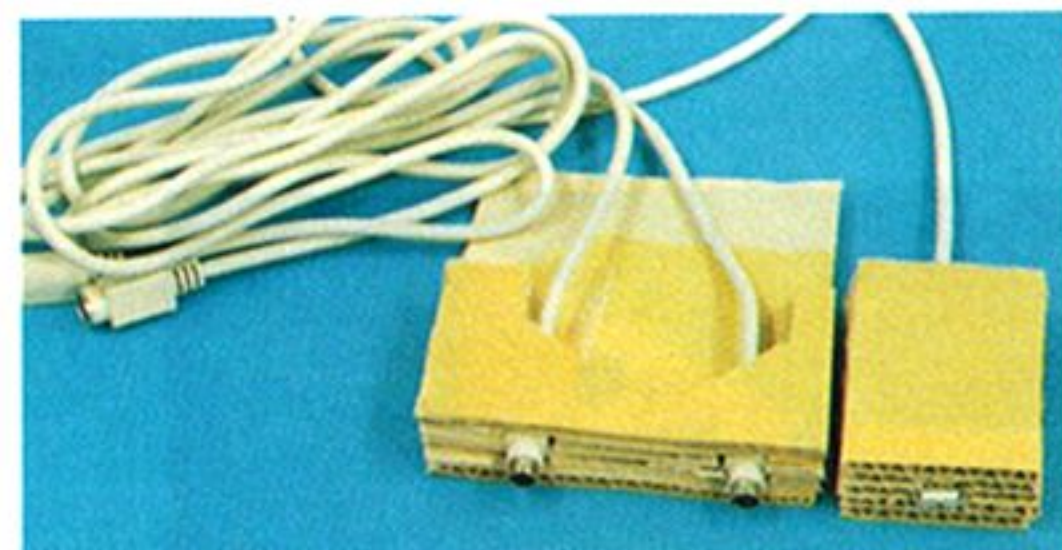
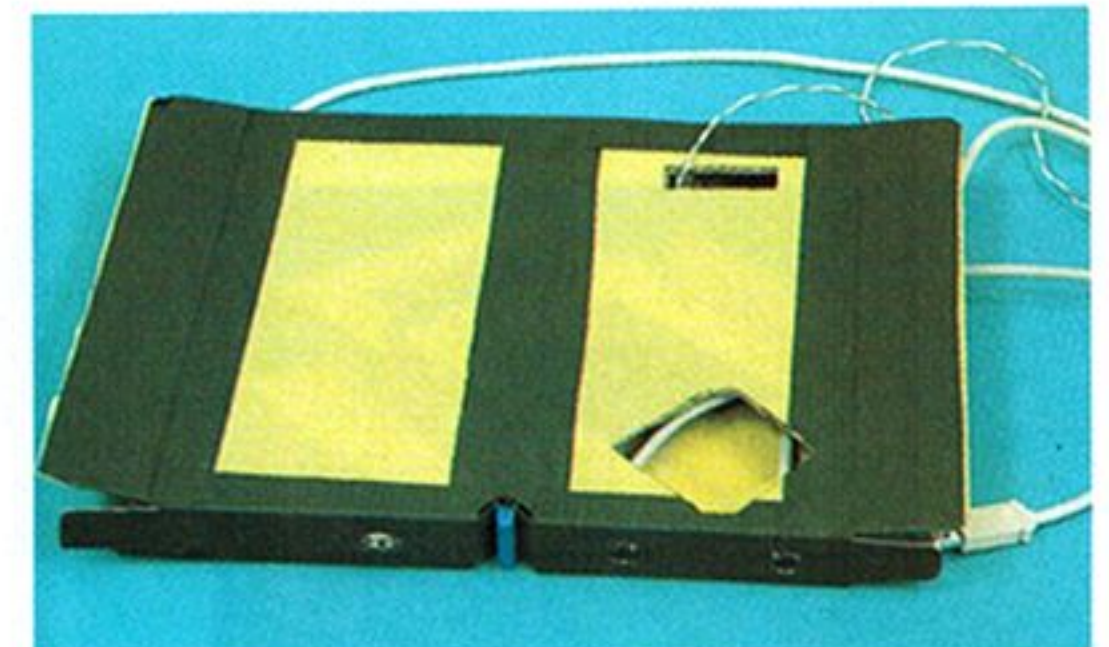
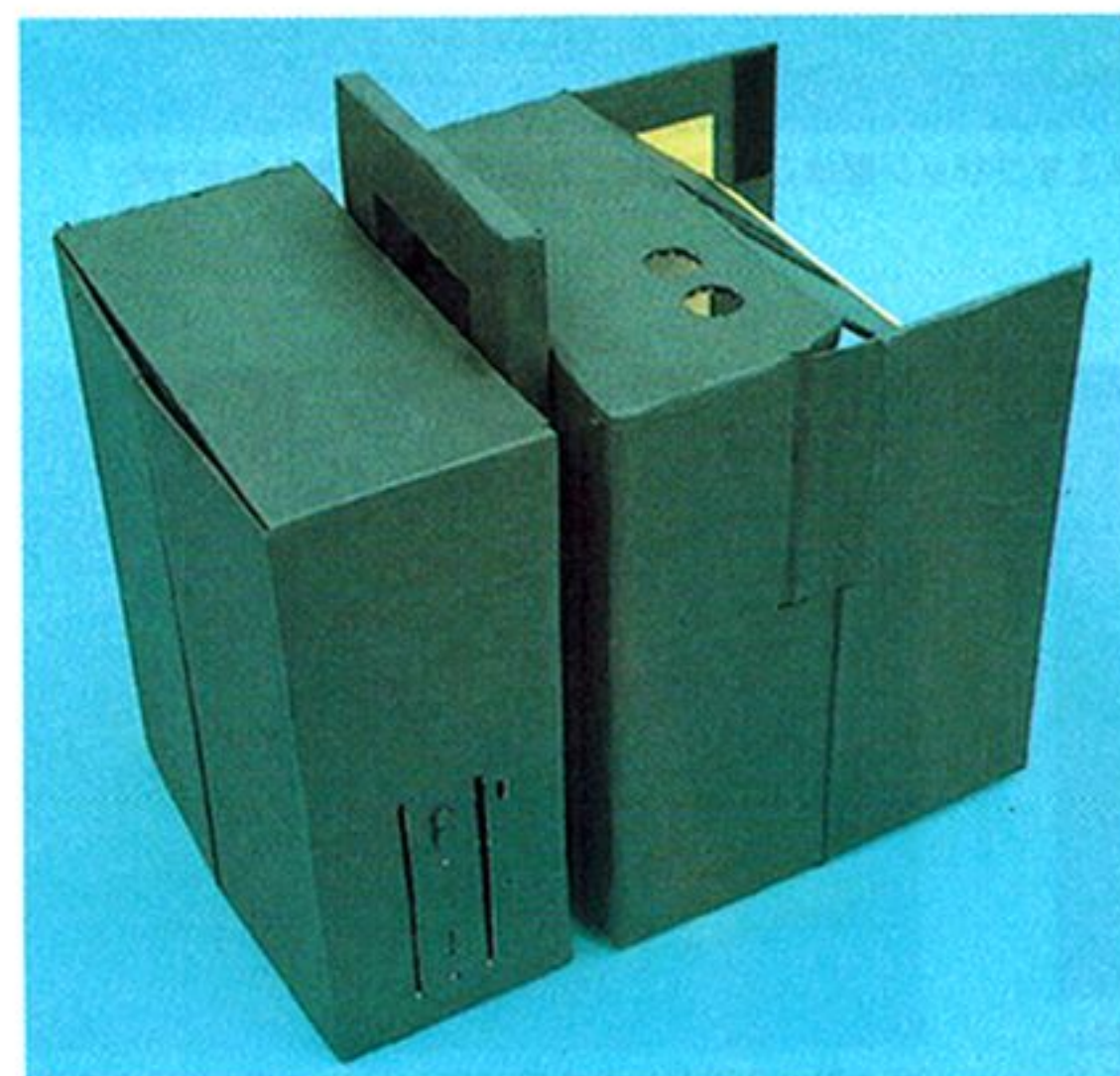
これはブリッジの受けとなる左側のタワー。手を切らないようにね

ブリッジ部に切れ込みを入れて，キャリングハンドルを通す。ちゃんとハンドルは上下に可動するぞ

右タワー上部につけるリセットとターボスイッチのユニット。押したときに抜けないように，しっかりダンボールで固める

とりあえずブリッジ部はこんな感じ。左右のタワーがつながった

リセットユニットを取り付けた右タワー。LEDがつく斜めの切り込み部分の加工にも注目。結構芸が細かい

最終的な右タワーの形状。背面にはコネクタ類とビデオカードの穴が開けられている。底面の穴はファン

左タワーに入るドライブユニット。スロットインのCD-ROMドライブとFDDが，フロントマスクを外されて固定されている

左タワーにスリットを開けて，ドライブユニットをセット。FDのほうも，スリットは5インチサイズ

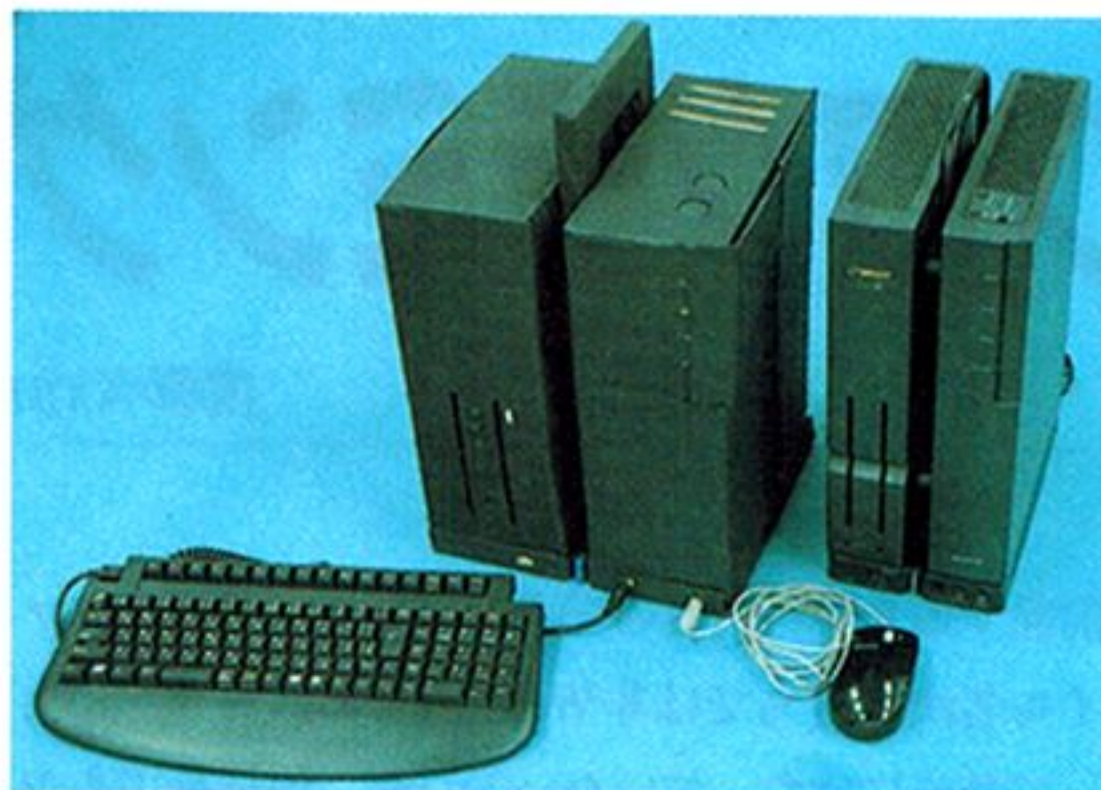
左タワーの内部。ドライブユニットの上には電源とHDDが乗っている。設計の甘さからHDDは斜めに1台のみ

ケーブル類を右タワーに出してみた。中央の穴はマザーボードで塞がれてしまうので，ハンドルとブリッジの隙間から出さねばならない。これも設計ミスだが，あとの祭り

市販の筐体用LEDは緑・黄・赤の3個セット。LEDはたとえつなぐものがなかろうと，どうしても4個ということで，別途青色LEDを使用する。頭が丸いものしかなかったので，なんとなく四角く加工

ダンボールの上から黒いラシャ紙を貼り付けていく

紙を貼り終わった右タワー。結構いい感じ

左タワーも同様に紙を貼り，ドライブのスリットを開ける

どうせ上のほうしか見えないけど，キャリングハンドルにもしっかりと

底部になるダンボール箱に切れ込みを入れ，電源スイッチを取り付ける。右上の穴は右タワーのファン用の穴

キーボード/マウスコネクタユニットとUSBコネクタユニット。それぞれ延長ケーブルで背面からフロント下部に持ってくる。抜けないようにしっかり固める

ユニットを固定し，紙を貼る。青い電源スイッチがポイントだ

ここで仮組みしてみる。なんかずいぶんらしく見えてきたぞ

タワー，底部を接続し，マザーなども収めてみた。延長ケーブルがちょっと長すぎて，右タワーの内部はかなりひどい状況

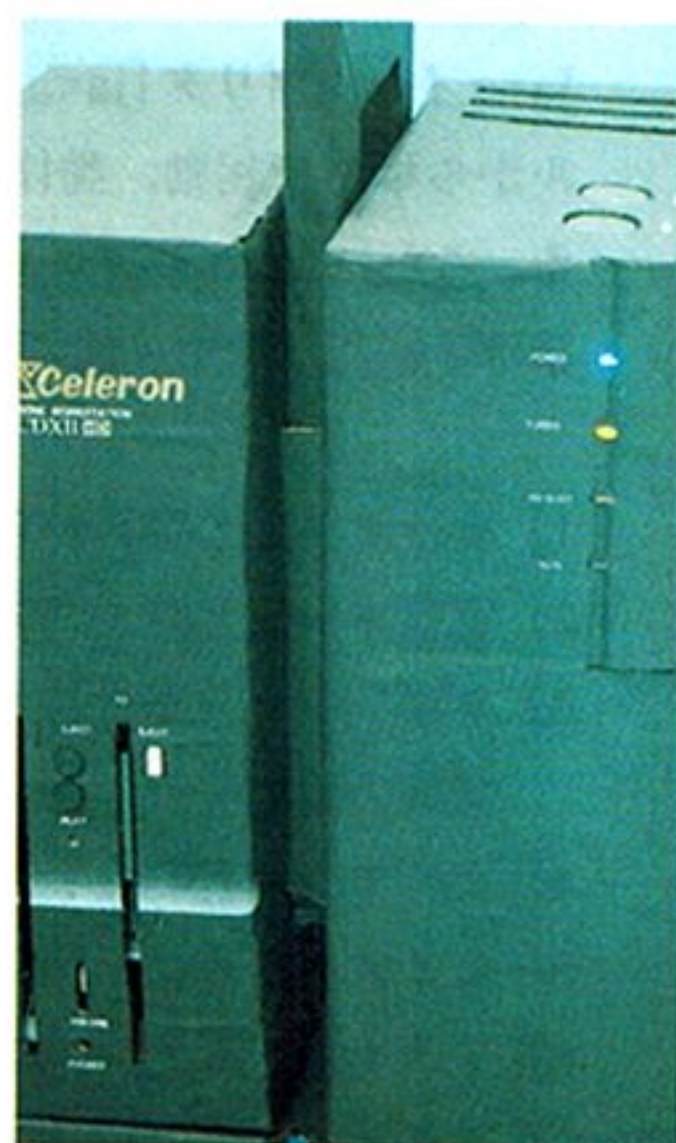
外形は一応完成。うんうん，上出来。底部背面の吸気孔がイカス。問題は動くかどうかだが

XVIと並べてみると，なんと偶然にも高さも奥行きもほぼ同じ。いやマジで。ちょっと肥満ぎみだけど

いよいよ文字入れ。プリントごっこのメッシュを直接当てて，白インクを乗せていく。結構あわせるのが大変だ

SD-X Celeron

ロゴは熱を加えると盛り上がるゴールドのパウダーで別途作成

ロゴ，文字の入ったSD-X Celeron。カッコイイ？ところどころ文字がずれているけれど，そこはご愛敬

絶対一発では動かないという意見が（U）氏と一致していたのだが，マウスとキーボード，IDEのマスターとスレーブが反対だったということを除いて，すんなり起動した。うひょー，ブラボー！　イカスー！

ゴールドのロゴもゴージャスに，ブルーのPOWER LEDがラブリー

もちろん側面には「あの文字」が

CD-ROMとFDはこんな感じで挿入される。ところで，へこんでるように見える？　見える？

ピンポ～ン

おにいさん：今日ご紹介するのは，このSD-X筐体です。

おねえさん：まあ，カッコイイ！

おにいさん：ちょっと太めのツインタワー，右にマザーボード，左にストレージ関係と電源を収めた，まさに未来のデザイン。

おねえさん：これならばお部屋のインテリアとしてもぴったりですね。

おにいさん：超軽量のダンボール素材を使用し，なんと重量は800グラムと，従来の筐体では考えられないほどの軽量化を実現。収納可能なキャリングハンドル付きで，持ち運びもらくらくです。

おねえさん：パソコンの重量って，半分以上は筐体の重さですものね。これならお掃除のときの移動も簡単。

おにいさん：さらに，筐体の左右はおごそかな観音開きになっていて，メンテナンス性も抜群。マジックテープで留めてありますので，ドライバレスで開閉が可能なんです。

おねえさん：それはクロックアップの強い味方ですね。

おにいさん：フロントにキーボード，マウス，USBコネクタを配し，POWER LEDは電源スイッチとおそろいのブルーのものを使用。ラシャ仕上げの表面に，ゴージャスな黄金のロゴプレートに加え，右側面には“POWER TO MAKE YOUR DREAM COME TRUE”の文字を入れてあるんです。

おねえさん：まあ，お洒落。誰もが夢見る筐体ですね。

おにいさん：本日はこのイカしたSD-X筐体を，なんと特別価格の19,800円でご奉仕。色はチタンブラックのみ。分割払いは月々1,980円の12回払いとなります。さらに，今お求めになると，350WのATX電源をプレゼント。お申し込みは今すぐDOS/V magazine編集部内Oh!X制作委員会まで。

おねえさん：お電話お待ちしてま～す。

うそ。売ってません。こんなもん大量生産できません。くれぐれも申し込みの電話などなさらないように。もっと簡単にできるかなーと思って，安易に作り始めたんですが，こんなに時間がかかるとは思ってませんでした。製作時間は30時間といったところですか。CD-ROMマスターアップ前の（U）氏にも多大な迷惑をおかけしまして，この場を借りてお詫び申し上げます。作った以上は使いますが，いちばんの問題は燃えやすいってことですかね。しかもクロックアップしてるし。あとは電磁波がバリバリ放出されててVCCI基準を満たさないってこともあるけど，それはたいした問題じゃないか。ま，作りたきゃ作ってもいいけど，なにかあっても責任は取れませんのでそのつもりで。

Javaアプリケーションと I/O プログラミング

霧雨/Kiriu

Javaというと Webでのアプレット用途しか知らない人も多いが，プラットフォーム非依存の開発言語ないし，実行環境として非常に有望なものである。ここではJavaをより実戦的に使うためのアプリケーション作成について解説する。

■Javaアプリケーションと I/O プログラミング

最近，インターネットではJavaアプレットのアニメーションやゲームを使用したアクティブなホームページをよく見かけるようになった。Javaの知名度も上がり，本職のプログラマ以外の方でも実際にアニメーションするJavaアプレットなどを作成することが増えてきている。だが，Javaアプレットで実際にゲームやビジネスアプリを作る際，Java特有の機能制限のために仕様実現が難しくなる場合がある。Javaアプレットをかじり始めたときに，必ず一度はぶつかる壁であろう。

だが，Javaで作られる実行形式には，Javaアプレット以外にJavaアプリケーションというものが存在することをご存じだろうか。実はJavaアプリケーションにはJavaアプレットのような制限がないので，これまでJavaでできないと思っていたことも実現できてしまうのである。Javaアプレットに比べ，いまひとつ知名度も解説書も少ないJavaアプリケーションだが，基本的なものはコンソールで動くC言語ライクな感じのプログラミングになるので，かえって取っつきやすいはずである。

また，Javaアプレットも署名という機能を使うことにより制限を取り除くことが可能なので，いままでJavaの制限がきついと思っていた方はぜひ最後まで読み進めてほしい。

ここでは，Javaアプレットは作れるという初級者向けに，JavaアプリケーションとJavaアプレットとの機能比較，ストリームの概念，およびI/Oアクセスの基礎的な方法を紹介する。また，サンプルはファイル＆URLアクセスを行うものを用意した。

なお，Javaは各バージョンで微妙に仕様が違うため，現在主流のJava Developers Kit (JDK) 1.1を対象とする。サンプルの開発はSymantec Visual Cafe for Java PDE v2.0J (JDK1.1.3) で行った。また，開発環境がWindows95であるため，一部解説にWindowsに特化した説明があることをご了承いただきたい。

■Javaアプレットと Javaアプリケーション

Javaアプリケーションにできること

Javaアプリケーションとは JDK1.1に付属の Javaインタプリタ「java」を使用して，コンソールから単体で起動，動作するソフトの形式である。これはJavaアプレットと比較して，以下に挙げる機能が使用可能になっている。

1) 実行マシン上のファイルへのアクセス
2) 実行マシン上のプログラム起動
3) ダウンロード元ホスト以外へのネットワーク接続の確立
4) 実行マシン上のライブラリの使用
5) ネイティブメソッドの定義
6) すべてのシステムプロパティへのアクセス

まず1)だが，Javaアプリケーションはハードディスク上のローカルファイルに対して，読み込みと書き込みが可能ということだ。Javaアプレットは Web上からダウンロードして実行するため，ダウンロードした先のファイルシステムに対し損害を与えるようなプログラムを阻止する目的で，この機能は使用できなくなっている。

2)も同様である。以前，悪質なサイトがインターネット上で騒がれたことがあったが，これはホームページにアクセスした瞬間，アクセスしたマシン上のプログラムを強制実行したり，Windowsがインストールされているフォルダを丸ごと削除するといったものであった。もっとも，ファイル名決め打ちでは非Windowsのユーザーにとって脅威にはなっていないのだが，他機種でもフォーマットをするツールなど，第三者に勝手に実行されては困るプログラムは存在する。こういうことができるとシステムに損害を与える可能性があるため，Javaアプレットでは使用できなくなっている。

次に3)だが，JavaアプリケーションはJavaサーブレット (サーバJavaアプレット) などと，どこにでもTCPやUDPでネットワーク接続を確立することが可能である。これに対し，Javaアプレットはダウンロード元とのみしか接続できない。URLを指定してほかのホームページにジャンプするという程度のことであれば，Javaアプレットでも可能である。

一応，ダウンロード元のサーバにJavaアプリケーションを置き，ダウンロードしたJavaアプレットが使う設定などの保存を行うようにすることで，1)のファイルI/O制限が間接的に外せる。ネット上でスコアを競うようなゲームなどで応用可能だろう。

4)は実行マシン上に存在するパッケージを使用することが可能という意味である。これに対してJavaアプレットでは，自分自身のソースコードとJavaアプレットを表示するビューア (ブラウザまたは，JDK1.1に付属するJavaアプレット表示ツール「appletviewer」) が提供する機能しか使用できない。つまり，ダウンロードされた先の表示環境で用意された機能に制限されてしまうということになる。

表1 システムプロパティ一覧

プロパティ名	Applet	プロパティの説明	筆者の環境の例
file.separator	OK	ファイル区切り文字	¥
java.class.path	NO	CLASSPATH	
java.class.version	OK	CLASS バージョン	45.3
java.home	NO	インストールディレクトリ	
java.vendor	OK	ベンダー固有文字列	Sun Microsystems Inc.
java.vendor.url	OK	ベンダー URL	http://www.sun.com/
java.version	OK	Javaバージョン	11
line.separator	OK	行区切り文字	(改行コード)
os.arch	OK	ＯＳ・アーキテクチャ	x86
os.name	OK	ＯＳ名	Windows 95
path.separator	OK	パス区切り文字	;
user.dir	NO	user作業ディレクトリ	
user.home	NO	userホームディレクトリ	
user.name	NO	userアカウント名	

筆者の実行環境：SONY VAIO505 EX/64，Windows95 OSR2.1，Symantec Visual Cafe for Java 2.0J，JDK1.1.3
補足：userホームディレクトリは，通常Javaコンポーネントが置かれている場所である。
userアカウント名は，Windows95の場合コントロールパネルのネットワーク設定ダイアログで設定されたユーザー情報の部分が反映される。

いい例は，2大大手ブラウザでJavaアプレットを表示したときの動作の違いだろう。読者の皆さんも，特定のJavaアプレットがどちらか片方のブラウザで動かないという経験をしたことはないだろうか。これら各ブラウザのJavaVMの違いについては後述する。

5)はネイティブメソッドの定義と統合である。これはJNI(Java Native Interface)という機能で，Java側で用意されたメソッドで目的にたえない場合，C言語などで記述された外部プログラムを呼び出すことが可能というものである。ほかの言語では当然なんでもできてしまうので，これもJavaアプレットでは使用できなくなっている。

6)のシステムプロパティというのは，Javaプログラムが実行する際の環境変数のようなものである。System.getProperty()メソッドでデータの参照を，System.getProperties()メソッドでリストの一覧を得られる。

参考として，表1にシステムプロパティの一覧と，筆者のWindows95マシン上で実行したときに得られた値の例を示す。この中でAppletという列でNOになっているものが，Javaアプレットがブラウザとappletviewer上での動作時に使用不可能なものである(念のため，筆者の環境での値もあえて表記していない)。前述したように，ダウンロードされたWebサイトとの通信だけは可能であるため，悪質なサイトがハッキングのためにユーザーの環境を探るということも十分に考えられるので，これも使用できなくなっている。

一応，すべてのシステムプロパティを表示するサンプル，Systems.classも用意してあるので，リストの中で表示されてない項目は，読者の皆さんがご自分の環境で実行し，どのような値が格納されるのかをご自分の目で確かめていただきたい。

なお，Javaアプレットで強引にこれらの機能を使おうとしても，セキュリティマネージャというセキュリティの監査役がSequrity.Exceptionを返してしまうため，まず正常に動作しない。のちほど実例でお見せするが，Javaコンソール上にエラー内容が表示され，場合によっては表面上固まったように見えてしまうのもある。こういったセキュリティの堅牢さはJavaアプレットの利点のひとつであり，かえってこれくらい厳しくないと，ネットワークビジネスには向かないだろう。

Javaアプレットにできること

なにかと制約が多いJavaアプレットだが，実はJavaアプレットでのみ使用可能な機能も，数は少ないが存在している。

1) 音声(サウンド)を使う
2) 同じWebのHTML上にあるほかのアプレットのメソッドの実行
3) ビュアとのコミュニケート

1)の音声の形式は，Sunが提唱するauというフォーマットにのみ対応している。この形式は主にUNIX上で使われることが多いが，JDK1.1では以下に示す形式しか扱えない。

●JDK1.1で使用可能なauファイルの形式
・圧縮方式：CCITT μ-law圧縮
・レート　：8kHz
・コード　：8ビット
・チャネル：mono

2)だが，同じ場所からダウンロードしたアプレットであれば，同じHTML上に表示されているほかのアプレットのメソッドを実行可能である。ただし，オブジェクト指向言語のルールどおりpublic宣言でないと呼び出しはできない。ほかにも，同じページ上に表示されたほかのアプレットを検索したり，アプレット間で通信を行うことも可能なので，ゲームなどでスコアの表示部分を別アプレットにするなどの応用方法もある。

3)だが，Javaアプレットが表示されているブラウザ，もしくはappletviewerと交信を行うことが可能である。現在表示されているHTMLや自分自身のパラメータを取得したり，ステータスウィンドウへメッセージを表示したり，指定したURLの内容をブラウザの好きなフレームに表示したりするようなこともできる(ちなみに，これはappletviewerでは無視される)。

Javaアプレットの例外と制限解除

これまでの解説では，Javaアプレットが制限だらけのようないい方をしてきたが，実は例外が存在する。ローカルファイル上からappletviewerを使用して読み出された場合は一部の制約がなくなってしまうのである(CLASSPATHが示す場所から読み込まれたアプレットであればすべての制約がなくなる)。

さらにもうひとつの例外として，「署名付きアプレット」というものがある。「JAR」および「署名」という機能を使用すると，実はこれまで述べてきた制限をすべて外すことができてしまうのである。これはWebサイトからダウンロードして即実行というJavaアプレットの利点と，一切機能制限のないJavaアプリケーションの利点の両特性をあわせ持つ。署名付きアプレットの作成手順は煩雑で概念も少々複雑であるが，ビジネス向けのJavaアプレットを作るには必須であろう。

まずJARだが，これは「Java ARchive」の略で，コンポーネントファイルをひとつにまとめるためと，圧縮してダウンロードに要する時間を短縮するために用意された機能である。ARchiveと名がつくとおり，ファイルの圧縮・解凍・まとめを行える。このJARというフォーマットは，インターネット上で広まっているzip形式を拡張したものと思ってもらえればよい。JDK1.1にはJARを扱うツールが標準添付されており，またそれを扱うパッケージ(java.util.zip/java.util.tar)も用意されているので，JARアーカイバなどのJavaアプリケーションをユーザーが作成可能である。

なお，コンソールから単体で使用可能な「jar」というJavaアプリケーションのツールがJDK1.1には標準で添付されている。これの基本的な使い方を説明しよう。JARファイルにするclassファイルをテンポラリフォルダなど，1カ所に集め，同じ場所で以下の内容を打ち込む。

```
>jar cf jarファイル名 *.class
```

cオプションは新規作成，fオプションは2つ目の引数でJARファイル名を指定するというものである。これでカレントにあったclassファイルがJARファイルとしてまとめられ，かつ圧縮される。ちなみに解凍するにはeオプションを指定すればいい。

なお，このJAR形式でJavaアプレットを圧縮した場合，HTMLの表記方法が少々異なる。実例は以下のとおりである。

```
メインアプレット名：Sample.class
JARアーカイブ名：Sample.jar
<APPLET ARCHIVE="Sample.jar"
CODE="Sample.class"></APPLET>
```

タグの中のARCHIVEにはJARファイル名を，CODEにはクラス名を指定する。サンプルではたまたまJARファイル名とクラス名が一緒になっているが，名称を一致させる必要はない。

また，Javaアプレットが複数ある場合にはCODEにメインクラス名を指定する。Javaでプログラムを作成すると，通常class単位でたくさんファイルが生成されるので，別段セキュリティを外す必要がなくても複数のclassファイルを1ファイルにまとめたいときには役に立つはずだ。

次に署名だが，これはPGPやMicrosoftのActiveXでも同じような概念を使っているので，耳にしたことのある方もおられるかもしれない。この機能を簡単にいうと「認証」ということになるだろう。

ダウンロードされたJavaアプレット，およびそのダウンロード元が信用に値するのであれば，別にJavaアプレットの機能制限はもともと必要がない。だが，そのためにはダウンロードされたJavaアプレットが，本当にその開発元が配布したもので，なおかつ改竄されていないという証拠が必要である。その証拠として機能するのが署名

である。要は直筆サインや銀行登録印の代わりのようなものだと考えればいい。

暗号化には，DSAもしくはMD/RSAという方式を使うことが可能で，データ鍵は512ビット長で暗号化される。また，署名の認定はX.509という方式が用いられている。暗号化技術は国家機密の扱いを受け，高いビット数を扱う最新技術は輸出規制をされるのだが，Javaに使われる技術は日本国内でも使用可能なので問題ない（たとえば電子メールの暗号化に使われるPGPでもアメリカ国内でのみ使用可能なものと，全世界で使用可能なものに分かれていたりする）。なお，実際にJARファイルに署名をつけるには，JDK1.1に標準で添付されている「javakey」というJavaアプリケーションツールを使用する。

署名付きアプレットを動かすと，セキュリティに関わる動作をした瞬間に，動作の判断をユーザーに求めるワーニングダイアログが表示される。Internet ExplorerのActiveXコントロールをダウンロードするときにも，認証を求めるダイアログが出るが，これと同じものだと思ってもらえればいい。ちなみに図1は，ローカルファイルにアクセスする署名付きアプレットを実行した場合に表示される例である。これはセキュリティに関わる動作ごとに表示され，ユーザーが許可した場合にのみ処理が行われる。この動作を平たくいえば，制限されている機能を使うときは警告を出すので，ユーザーが最終的に判断してほしいということになる。ダイアログが出るというのが少々面倒かもしれないが，Javaアプリと同じ機能が使えるというのは魅力だろう。

大手ブラウザにおける互換性の問題

先ほど，Javaアプレットがローカルファイルから読み込まれた場合の例外を解説したが，実はInternet Explorer3.0x以降に付属する「jview」から読み込まれた場合も，appletviewerとほぼ同様な制約の解除がある。「jview」はオプションの設定でJavaアプリケーションもJavaアプレットも表示可能だが，Internet ExplorerのMicrosoft VM for Javaを使用するので，完全にJDK1.1互換とはいかないため，注意が必要である。これは，現在SunとMicrosoftが争っているため，今後もどうなっていくかわからない。

図1

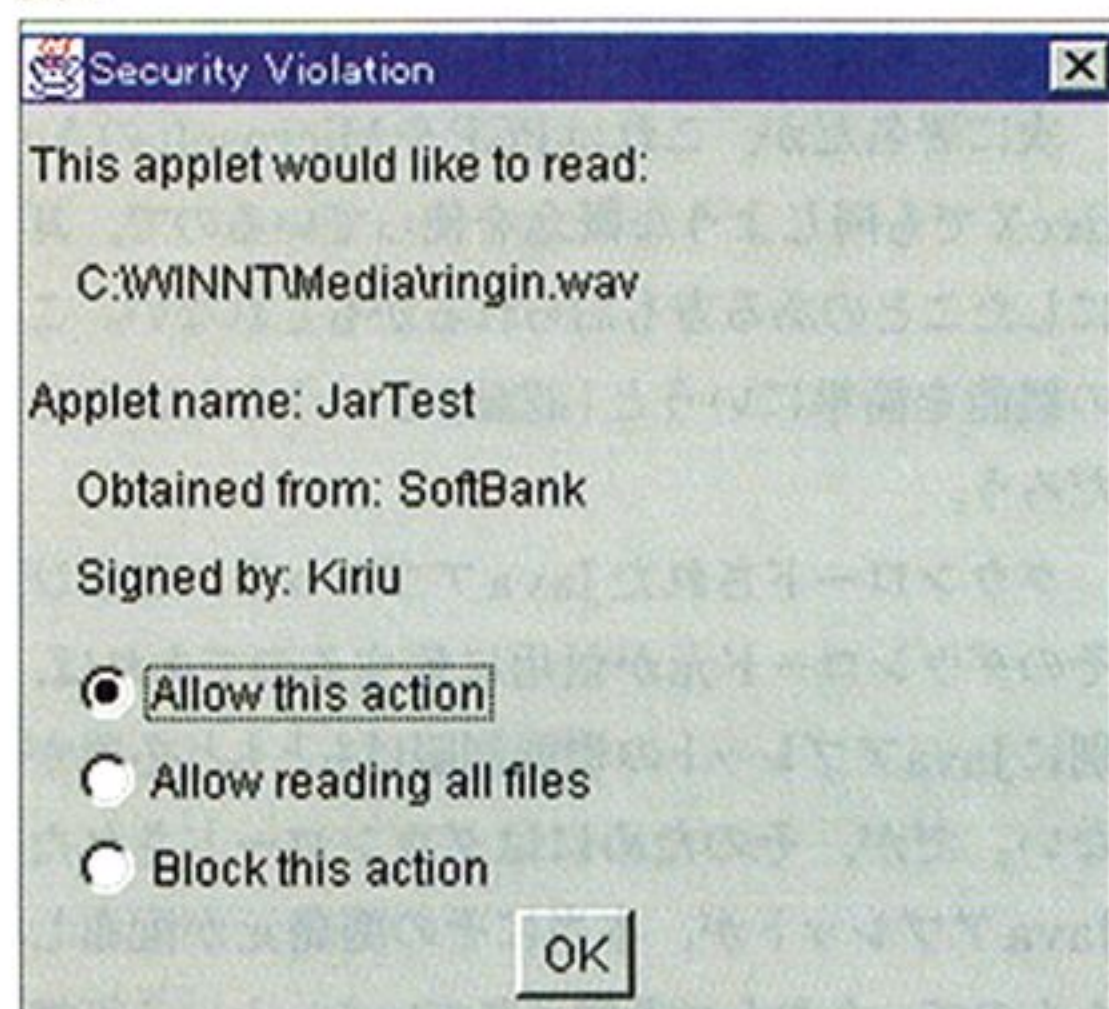
Security Violation
This applet would like to read:
C:\WINNT\Media\ringin.wav
Applet name: JarTest
Obtained from: SoftBank
Signed by: Kiriu
◉ Allow this action
○ Allow reading all files
○ Block this action
OK

ローカルハードディスクにアクセスしようとしたときに出るセキュリティダイアログ

表2　大手ブラウザにおけるJDK1.1非互換機能

InternetExplorer 4.0x
・RMI（※1）
・JNI（※1）
・署名付きアプレット
※1　この機能に対するJDK1.1対応パッチが下記アドレスから手に入る。

HomePage　http://www.microsoft.com/msdn/news/rmijni.htm
パッチ本体　ftp://ftp.microsoft.com/developer/MSDN/UnSup-ed/rmi.zip

Netscape Communicator4.0x
・JavaBeans
・AWT（※2）
・署名付きアプレット
※2　4.03/4.04用のパッチのほか，AWTやJavaBeansなどに対応した4.05pro-us版（本体と一体化している）というバージョンも存在する。これも下記アドレスから入手可能。

HomePage　http://developer.netscape.com/software/jdk/download.html
Netscape4.03用（Win32）
ftp://ftp.netscape.com/pub/communicator/smartupdate/english/windows/windows95ornt/awt/403wt.zip
Netscape4.04用（Win32）
ftp://ftp.netscape.com/pub/communicator/smartupdate/english/windows/windows95ornt/awt/404awt.zip
Netscape4.05pro-us版（Win32）
ftp://ftp.netscape.com/pub/communicator/4.05/development/english/windows/windows95_or_nt/4.05_plus_JDK1.1_pr1/cp32e405.exe

Netscape Communicatorも残念ながらJDK1.1の完全互換とはなっていない。参考として，表2にInternetExplorerとNetscapeCommunicatorにおける，JDK1.1に非互換の機能一覧を挙げておく。この記事のサンプルを試す場合やご自分でJavaアプレットを作成する場合などでは注意していただきたい。一応各社からJDK1.1対応パッチが無償で提供されているのだが，いずれも部分的であり完全ではないという状況である。

では，これら2大ブラウザを完全にJDK1.1に対応させるにはどうするかというと，Java Plug-in（Java Activator）というものがSunから提供されているので，それを利用する。このプラグインをインストールすると，新しくJDK1.1対応JavaアプレットのMIMEタイプが追加され，HTMLのタグ表記を変更することで，完全な互換性を得られる（現在の最新バージョンは1.1.1）。ちなみに<APLLET>タグ表記だが，Sunが無償で提供している「HTMLコンバータ」で簡単に変換することが可能だ。

先ほど述べた署名付きアプレットの仕様も，Sun純正のJavaブラウザであるHotJavaと，ほかの一般的なブラウザの種類によってかなり仕様が異なり，署名をつけてもそのままでOKというわけにはいかない。

InternetExplorerは，Javaアプレットを圧縮および，まとめるフォーマットとしてMicrosoft独自のcab圧縮という形式を使用する。このフォーマットはこの会社が販売する製品，つまりWindows95などのCDに，インストール前の状態で保存するときに使われるもので，Visual J++などに付属している。ちなみに署名のつけ方そのものも，ブラウザのバージョンが3.0xか4.0xによっても微妙に異なるほか，実行時に認証が必要となり，認証サービスサイトが必要となる。

Netscape Communicatorは，JAR形式までは純正と一緒だが署名のつけ方が異なる。セキュリティがかかっている機能を個別に外すことは可能だが，ソースコードに特定の宣言を記述する必要がある。こちらも，認証はInternetExplorer同様，専用サイトが必要となる。

これらの違いに対処するには，ブラウザ別に署名付きアプレットを作成してページを分割するという方法と，Java Plug-inをインストールするという方法がある（各社提供のJDK1.1パッチには，残念ながらこの機能の対応は含まれていない）。どちらもHTMLを変更する必要があるが，現時点では後者が最良の方法と思われる。

とかくブラウザのJava動作環境は現在でも混沌としているが，筆者としてはなるべくSun純正の方式で作成することをおすすめする。

■実例とストリーム

コーディングと動きの違い

JavaアプリケーションとJavaアプレットの動きの違いを比較するために，基礎的なサンプルSample.classを用意した。Sample-1はJavaア

プリケーション版，Sample-2は比較のために用意したJavaアプレット版，Sample2.classである。Javaアプリケーションの動かし方は簡単である。classファイルをパスの通った場所かカレントに置き，コンソールから以下のように打ち込めばいい。

```
>java メインクラス名
```

紹介したサンプル，Sample.classの場合なら，このように打ち込む。

```
>java Sample
```

JavaアプレットタグをHTML記述する場合同様に，大文字小文字がきちんと認識されるので，間違えないようにしなくてはならない。また，拡張子の「.class」は打ち込んではいけない。Javaにおいてピリオドはパッケージの区切りを示すので，この場合はクラスの名称のみを指定する。

打ち込みに間違いがなければ，Javaインタプリタが起動し，同じコンソール上に以下のようなメッセージが表示される。

```
Sample Java Application
```

次にJavaアプレット版を実行してみよう。Sample2.htmを手持ちのブラウザで表示すればよい。だが，Javaの領域らしいエリアは表示されるが，メッセージはどこにも出てこないはずだ。

このサンプルのメッセージ出力はSystem.in.printin()という標準出力メソッドを使用しているが，これはJavaコンソールに表示されるのである。ブラウザが持っているJavaコンソールを表示して確認していただきたい。

また，appletviewerもJavaインタプリタ同様，コンソールから入力して起動するが，こちらでもコンソールのほうへメッセージが表示される。ちなみに文字をアプレットの領域上に表示するのは，java.awtパッケージのlabelやtextFieldなど，GUIコンポーネントで行うのが一般的だろう。これは今回の解説から外れるが，JavaアプレットでもJavaアプリケーションでも使える。

2つのサンプルのコードを比較して，違う点をざっと見てみよう。まずJavaアプリケーションのclassの定義は，以下のようになっている。

```
public class Sample {
```

続いて，Javaアプレット版は以下のとおりになっている。

```
public class Sample extends Applet {
```

Javaアプリケーションがなにからも継承（extends）していないのに対し，JavaアプレットはAppletクラスから継承している。

Javaアプレットは基本的にブラウザ内で動作するのだが，この場合，Javaアプリケーションはコンソール上で動作する。これがフレームウィンドウを生成するJavaアプリケーションなら以下のようにフレームクラスから継承することになる。

```
public class Sample extends Frame {
```

ちなみにJavaアプレットでもFrameウィンドウを作成することは可能である。

次に，C言語で見慣れたmain関数が存在するのが目につくだろう。コンソールからC言語のように引数を得ることが可能となっている。C言語のようにargs[]というStringの配列が宣言されているが，args[0]にひとつ目の引数，argc[1]に2つ目の引数，以下同様に要素数の分だけ格納される。引数の最大数は，args.length()で参照すればよい。また，この引数を格納するString配列は，たとえプログラムのほうで使用しなくても必ず記述しなくてはならない。ただ，宣言するString配列の名称は別にargsでなくてもかまわない。

これに対しJavaアプレットには以下に示すイベントが存在する。

init()	初期設定など
start()	動作のメインとなる処理
stop()	一時停止時など
destroy()	リソース解放など

これらは仮想（virtual）関数なので，必要に応じて書き換えて使用する。逆に必要がなければデフォルトの内容が実行されるので，ソース中にこれらのイベントを記述する必要はない。

構造の簡単なプログラムであれば，init()およびstart()イベントに書いてある記述をmain()に移し換えることで，簡単にJavaアプレットをJavaアプリケーションに書き換えることも，またその逆も可能である。

ストリームを理解する

基本構造の違いの次は，ファイルI/Oの方法に移るが，実例を見る前に理解しておかなくてはならないことを解説しよう。それは「ストリーム」というものである。これはJavaでI/Oを行う際に必須の概念であり，クラス化されている。

Sunのホームページにある Java 入門書では「ストリームは流れている文字の並びである」と記されている。ストリームという単語は直訳すれば確かに「流れ」だが，この表現ではいささかわかりにくいと思う。概念そのものはC++言語の頃から

リスト1　Javaアプリの例

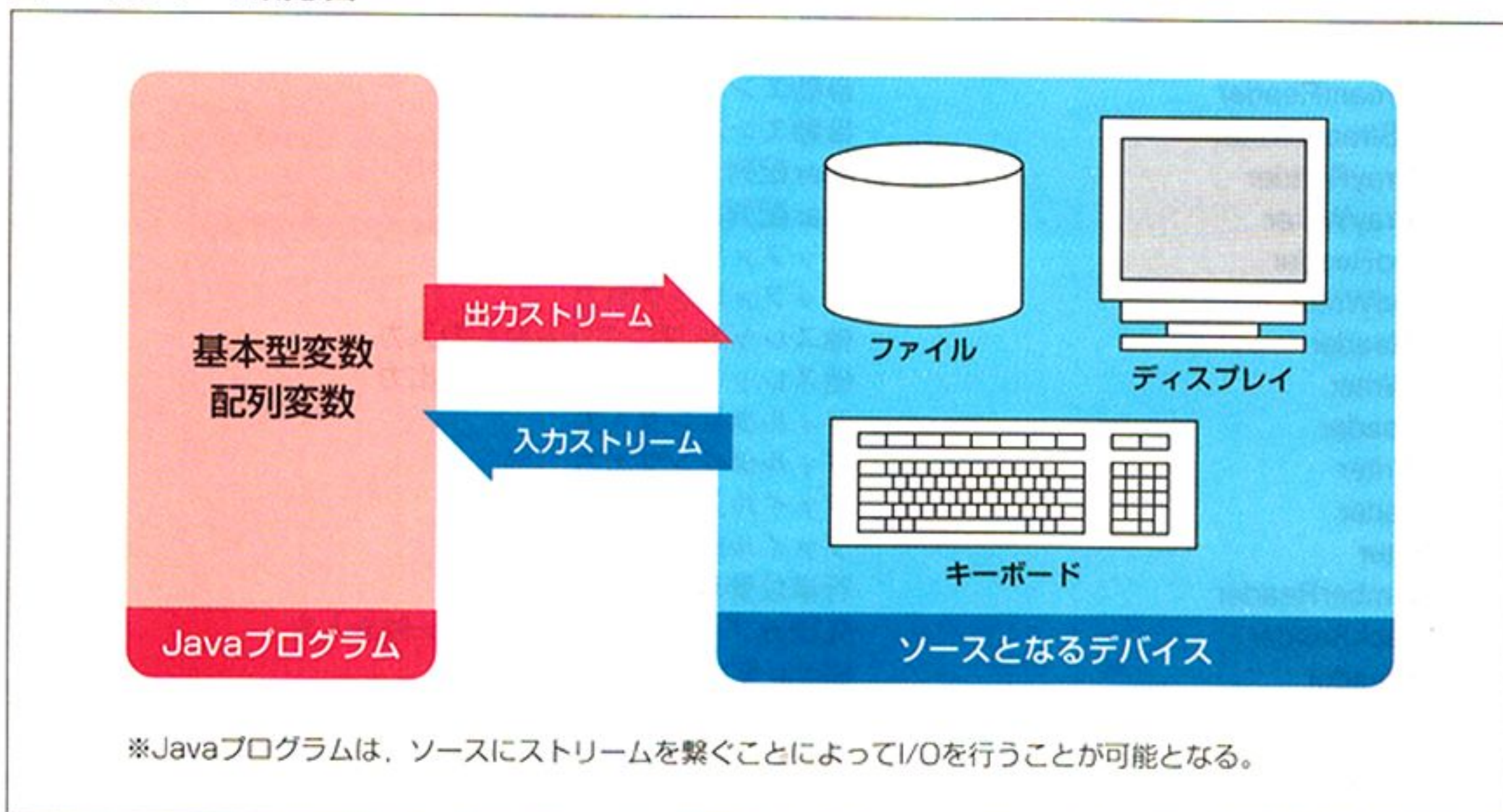
```
//
// 基礎的なJavaアプリケーション Sample.class
//
public class Sample {
      public static void main(String args[]) {
            System.out.print("Sample Java Application");
      }
}
```

リスト2　Javaアプレットの例

```
//
// 基礎的なJavaアプレット Sample2.class
//

import      java.applet.*;

public class Sample2 extends Applet {
      public void init() {
            // 初期化
            setLayout(null);
            setSize(200,200);
      }
      public void start() {
            // アプレットスタート
            System.out.print("Sample Java Application");
      }
}
```

図2　ストリーム概念図

基本型変数
配列変数
Javaプログラム
出力ストリーム
入力ストリーム
ファイル
ディスプレイ
キーボード
ソースとなるデバイス

※Javaプログラムは，ソースにストリームを繋ぐことによってI/Oを行うことが可能となる。

取り入れられているので，それほど目新しいものではないが，初めて聞くとなかなかピンとこないものではある。

ではストリームとはいったいなんなのか。それは，デバイスなどのデータ入出力の流れを抽象化したものである。

抽象化したといってもさして難しく考える必要はなく，画面，キーボード，ファイルなどへの入出力を行う場合に，ストリームを使うのだと考えればいい。図2はストリームの概念をイメージ化したものだが，この図のとおり，デバイスとJavaプログラムとの橋渡し役ととらえてもいい。たとえば，お馴染みの標準入出力もストリームである（ちなみにJavaではSystem.inとSystem.outというクラスがこれに対応する）。

実際のプログラムでは，用途に応じたクラスのオブジェクトを生成し，そのメソッドを呼んでデータの読み込みもしくは書き込みを行うというのが基本的な手順になる。

このストリームを扱うクラスは目的に応じて多岐にわたっており，java.io.＊パッケージ以外にも多数存在している。参考として，表3にjava.io.＊に含まれるストリームクラス名の一覧を示す。

ストリーム化の利点と欠点

少々大ざっぱな説明だったが，ストリームの概念がご理解いただけただろうか。ここでさらにもう一歩踏み込んで，なぜJavaではI/Oが「ストリーム」という概念で抽象化されているのか，ということにも触れておこう。

普通，機種やOSが変わると，デバイスへのI/O方法が異なる。プラットフォームが限定されていないJava上で，それらのドライバや制御ルーチンを逐一コーディングすると，プログラマの負担はとてつもないものになってしまう。それでは開発作業も非効率であるうえに，プラットフォームに依存しないというJavaの利点も失ってしまうことになる。

だが，I/Oがストリーム化されていれば，内部的な違いはJavaVMが吸収してくれるため，プログラマは機種やOSの違いを気にすることなく，I/Oルーチンを記述することができるのである（これはJavaやI/Oに限ったことではなく，なんらかの標準化を行おうとした場合，必ずついて回る問題でもある）。

このように，I/Oの抽象化はJavaにおいて非常に理にかなっているのだが，その性質ゆえに別の制限をもたらすことにもなる。機種，OS間で共通して存在するようなデバイス（たとえばディスプレイやキーボード）に関しては問題ないが，発売元が提供するデバイスドライバを必要とするようなデバイス機器は使用不可能となってしまう。その部分のみJNI（Javaネイティブインタフェイス）でC言語などで作ったネイティブコードを記述する手もあるが，100% Pure Javaの利点が活かせなくなる。

Javaが標準化するにつれ，ネイティブのAPIで作成したアプリケーションをJavaに移植するというパターンが増えてくると思われるが，製品の仕様によってはこの問題でかなり四苦八苦するだろう。これは，今後Javaのバージョンアップに期待するしかないというのが現状のようだ。

なお，部分的にネイティブコードを使った実際の例として，Novita Communications, Inc.社から発売されているNOVITA MAIL（ノビタメール）という製品を紹介しよう。これはHotJavaをベースに作られたJavaアプリケーションのインターネットメーラーだが，TWAINデバイス機器（デジカメ，スキャナなど）から画像を取り込む機能を有している。JavaにはTWAIN機能に対応したパッケージがないので，この部分のみをWindowsのネイティブメソッドで記述しているらしい（そのルーチンはDLLになっている）。そのため，この製品はWin32用という位置づけになっているようだ。Javaのパッケージが揃いきっていない現在では，このような応用も可能という例である。ちなみにこの製品は日本語化されており，Intercom, Inc.から市販パッケージとして販売されている。トライアル版もあるので，興味のある方は触ってみるといいだろう。

ファイルアクセスを行うには

ここからストリームを使用した例を見ていこう。

JavaアプリケーションでファイルI/Oを行うにはjava.io.＊パッケージを使えばよい。ストリ

表3 java.io.＊のストリームクラス一覧

■byte単位のストリームクラス

クラス	説明
InputStream	byte入力の基本クラス
OutputStream	byte出力の基本クラス
BufferedInputStream	バッファリングによる効率化入力
BufferedOuputStream	バッファリングによる効率化出力
ByteArrayInputStream	byte配列入力
ByteArrayOutputStream	byte配列出力
DataInputStream	Java基本型データ入力
DataOutputStream	Java基本型データ出力
FileInputStream	ファイル入力
FileOutputStream	ファイル出力
FilterdInputStream	フィルタリング入力
FilterdOutputStream	フィルタリング出力
LineNumberInputStream	行単位管理入力
LineNumberOuputStream	行単位管理出力
ObjectInputStream	オブジェクトのエンコード化(シリアル化)入力
ObjectOutputStream	オブジェクトのエンコード化(シリアル化)出力
PipedInputStream	他スレッド/プログラムからの入力
PipedOutputStream	他スレッド/プログラムへの出力
PushbackInputStream	先読みプッシュバックバッファ付き入力
PrintStream	印刷
SequenceInputStream	複数ストリームをひとつにまとめる

■char単位のストリームクラス

クラス	説明
Reader	char入力の基本クラス
Writer	char出力の基本クラス
InputStreamReader	自動エンコード入力
OutputStreamWriter	自動エンコード出力
CharArrayReader	char配列入力
CharArrayWriter	char配列出力
BufferedReader	バッファリング入力
BufferedWriter	バッファリング出力
PipedReader	他スレッド/プログラムからの入力
PipedWriter	他スレッド/プログラムからの出力
FilterReader	フィルタリング入力
FilterWriter	フィルタリング出力
FileReader	ファイル入力
FileWriter	ファイル出力
LineNumberReader	行単位管理入力
PushbackReader	先読みプッシュバックバッファ付き入力
StringReader	Stringをソースとする入力
StringWriter	Stringをソースとする出力

※読み込みと書き込みで対になっていないクラスもある。また，これ以外にもいくつかストリームクラスが存在するが，JDK1.1で推奨されないものは省いてある。

ームクラスからファイルI/Oに適したものをオブジェクト生成したあと，メソッドで入出力を制御し，close()メソッドでファイルを閉じるという手順を踏む。その際，try{}，catch{}文による例外処置を記述しなくてはならない。try{}に実際の処理を記述し，catch{}文には，try{}内で発生しうるエラーに対応するException(例外)と，エラー発生時の処理を記述する。

Sample-3は基本的なbyte単位のストリームクラスを使用し，ファイルのコピーを行うサンプル，Copy.classである。このサンプルは入出力をbyte単位で行ううえに，使用するクラスはバッファリングを行わないので，もっとも効率が悪いが，初歩的なサンプルということでご容赦いただきたい。

簡単に解説すると，まずFileInputStreamとFileOutputStreamのオブジェクトを生成し，コンソールの引数から取得したファイル名を渡す。FileInputStreamのread()メソッドはbyte単位で読み取りを行い，ファイルがEOFに達すると−1を返すため，while()文でループさせる。終了したら，close()メソッドでファイルを閉じる。

例外処理は3つ。まずはFileNotFoundExceptionで，ファイルが存在しなかった場合。IOExceptionはファイルの読み込み時になんらかの要因でファイルが読み書きできなかった場合。Exceptionはそれ以外のエラーが発生した場合である。サンプルでは例外が発生した場合でも特に処理は行わず，メッセージを標準出力に表示し，終了するのみとなっている。

リスト4，5は基本的なchar単位のストリームクラスを使用し，標準入力から受け取ったテキストファイルを読み取って標準出力に表示を行うサンプル，Type.classである。先ほどとは違い，ReaderとWriterというストリームクラスを使用している。byteとcharが別のクラスになっているのは，charの長さが2バイト(16ビット)と定義されているためである。これらのクラスを使用すると，特定の文字コード体系(JIS，EUCなど)や言語(英語などの1バイトコード圏や日本語などの2バイトコード圏の差)にとらわれないプログラムが容易に作成可能になっている。JDK1.0.2の頃は日本語処理に問題があったのだが，JDK1.1以降では解決したようだ(ただし，Sunが推奨するクラスとAPIを使用することが条件である)。テキストファイルや文字列を扱う場合などは，Reader，Writer系のクラスを，特に効率的なバッファリングを行ってくれるBufferedReader，BufferedWriterなどを使うことをおすすめする。もし，読み書きするコード体系が判明していて，かつ特定のコードを使いたいような場合は，InputStreamReader，OutputStreamWriterを使うとよい。

Type.classも簡単に説明しておく。これはコンソールからの引数によって，処理を分岐させるようになっている。引数が足りない場合はUsageを表示するのみにし，引数があればそれをFileReaderのコンストラクタに渡す。read()メソッドでcharの配列にデータを読み込む。ファイルがEOFに達すると−1を返すため，while()文でループさせる。終了したら，close()メソッドでファイルを閉じる。例外処理は先ほどのサンプルと同様である。

実はこの例も，逐一byteを読み込んでは文字に変換するという，あまり効率がよくない内部処理をしている。それを改善するには先ほどのBufferedReader，BufferedWriterクラスを使うとよい。以下がその例である。

```
BufferedReader inReader = new Buffered
Reader(new FileReader(ファイル名));
```

このように，Bufferedクラスでラッピングすると，I/Oのバッファリングが行われ，内部のデータ処理効率がアップする。

ちなみにファイルを扱うクラスはストリーム以外にも存在する。たとえば，java.io.Fileパッケージを使用すると，ファイルシステムを管理することが可能である。

このクラスのメソッドには，ファイルのアトリビュートを取得するものと，ファイルおよびディレクトリの作成，削除を行うものが含まれている。ファイルストリームでI/Oを行う際は，Fileクラスも併用してファイルの存在チェックをしたり，上書きチェックなどを行うほうが望ましいだろう。

ネットワークアクセスを行うには

JavaアプリケーションでネットワークI/Oを行うには，java.net.＊パッケージを使う。使用可能なのはTCPとUDPプロトコルである。また，URL通信を行うクラスもあり，URL上のHTMLファイルなどを簡単に取得することが可能である。

JavaはBSDと同等のネットワーク機能を持っており，ソケット(Soket)インタフェイスという独特の方法でアクセスを行う。「ソケット」とは，文字どおりLANケーブルや電話口など，現物のソケットを抽象化したもので，ネットワークの終端を表している。これもクラス化されており，入出力ストリームのメソッドを持つので，実際にはソケットオブジェクトを作り，そのメソッドで制御を行うことになる。

TCPプロトコルはデータの信頼性が高く，回線の切断を注意する程度で問題なく動作するものが作れる。また，ソケットクラスはI/Oストリームのメソッドを持っている。ファイルI/Oの基本ができていれば比較的簡単にプログラムすること

リスト3　Sample3.java

```
//
// URLのテキストを取得するサンプルJavaアプリケーション Sample3.class
//

import     java.lang.*;
import     java.io.*;
import     java.net.*;

public class Sample3 {
     public static void main(String args[]) {
          if (args.length == 0){
           System.out.println("[Usage] java Sample3 URLorIPaddress");
          }
          else {
               try {
                    int data;
                    // URLソケットオブジェクト生成
                    URL urlObj;
                    urlObj = new URL(args[0]);
                    DataInputStream inStream;
                    // OpenStream()はurlへ接続し、InputStreamを返す
                   inStream = new DataInputStream(urlObj.openStream());
                    FileOutputStream outStream;
                    outStream = new FileOutputStream("getdata.htm");
                    // ストリームから符号なしデータをByte単位で読み込む
                    for (;;){
                         data = inStream.readUnsignedByte();
                         outStream.write(data);
                    }
               }
               catch(EOFException e){
                    // ファイルがEOFに達した場合
                    System.out.println("url download.");
               }
               catch(IOException e){
                    // I/Oエラーが発生した場合
                    System.out.println("I/O Error.");
               }
               catch(Exception e){
                    // それ以外のエラーが発生した場合
                    System.out.println("Error....");
               }
          }
     }
}
```

リスト4　Type.java

```
//
// 標準入力から得たファイル名を表示するJavaアプリケーションサンプル Type.class
//

import     java.io.*;

public class Type {
     public static void main(String args[]) {
          try{
               FileReader inReader;
               char dataStr[] = new char[255];
               if (args.length > 0) {
                    // charストリームオブジェクト生成
                   inReader = new FileReader(args[0]);
                    // EOF(-1)を返すまでループする
                   while (inReader.read(dataStr) != -1) {
                         System.out.print(dataStr);
                    }
                    inReader.close();
               }
               else{
                    // 引数がない場合、文法を返す
                    System.out.println("[Usage] java Type filename");
               }
          }
          catch(FileNotFoundException e) {
               // ファイルが見つからない場合
               System.out.println(args[0] + " NotFound.");
          }
          catch(IOException e) {
               // I/Oエラーが発生した場合
               System.out.println("I/O Error.");
          }
          catch(Exception e) {
               // それ以外のエラー
               System.out.println("Error...");
          }
     }
}
```

リスト5　Type2.java

```
//
// 標準入力から得たファイル名を表示するJavaアプリケーションサンプル Type.class
//

import     java.io.*;

public class Type2 {
     public static void main(String args[]) {
          try{
               File inFile;
               inFile = new File(args[0]);
               System.out.println("Time=" + inFile.lastModified());
          }
          catch(Exception e) {
               // それ以外のエラー
               System.out.println("Error...");
          }
     }
}
```

が可能だろう。

UDPプロトコルは単純かつ高速であり，データの小さな処理に向いている。これは「データグラム方式」と呼ばれる方式で，パケット単位で通信を行うが，データ消失の危険性がある。プログラムする際には再送処理などで，少々高度なプログラミングを要する。

URLプログラムは読み込み先のContent-Type（データ形式）が判明していれば，手順は簡単だが，ブラウザのようなマルチスレッドと複数Contentを扱うと非常に高度になる。ちなみにURLStreamHandlerクラスを用いると，ftpやgopherなどのプロトコルにも対応したプログラムを作成可能である。

Sample-5は，URLクラスを用いてコマンドラインに指定されたIPアドレスもしくはホストネーム上のテキストを取得するサンプル，Urlget.classである。

ちなみにIPアドレスというのはネット上に繋がっているマシンに割り振られる固有の番号で「202.234.14.173」のように0～255までの3桁の数字を4つ並べ，ピリオドで区切ったものである。ネット上に繋がれるマシンはこれで固有識別が行われるが，どうしても人間が覚えやすい形式とはいえないので，DNS（ドメインネームサーバ）というネーム管理サーバが，インターネットアドレスとしてよく見かける「http://www.softbank.co.jp」のようなホストネームに変換してくれるのである。これが設定されたサーバは，世界中どこにいてもドメインネームでアクセスが可能となる。

余談として，ローカルファイルの場合の指定方法も見ておこう。たとえば，Cドライブ上のAUTOEXEC.BATを指定する場合はこのようになる。

```
file:///C:/autoexec.bat
```

ファイルの次にスラッシュが3つつくのでご注意いただきたい。ネットワークツールでも，このような指定方法で，ローカルファイルを扱える。

このUrlget.classも簡単に解説する。実行時の引数からアドレスを取得し，URLを扱う「URL」というクラスのオブジェクトを生成する。そしてそのコンストラクタには取得したアドレスを渡す。次にDataInputStreamオブジェクトを生成し，これにURLオブジェクトを繋ぐ（このストリームクラスにほかのストリームクラスを繋ぐという手法は次項で解説する）。

URLオブジェクトのOpenStream()メソッドは，実行すると実際にurlと接続を確立し，入力を得るためのInputStreamクラスを返すようになっている。取得したデータはローカルのgetdata.htmというファイルに保存する。こちらはシンプルなFileOutputStreamで開く。あとは無限ループで符号なし8ビットデータを読み込み，データを読み込みmファイルへ書き出す。終了はEOFExceptionでcatch}}し，その他の例外は，ファイルストリーム同様に行う。

実際に操作してみよう。コンソールから以下のように打ち込めばいい。

```
>java Urlget http://www.yahoo.co.jp
```

実行すると引数で指定したアドレスへ接続を確立し，HTMLファイルをローカルのgetdata.htmというファイルに保存する。上記の実行例ではHTMLファイルまでの記述がないが，インターネットサーバはデフォルトのファイル名が決まっているので，上記のような場合ではYahoo！のトップページが取得できる。

処理が終わったら，実行したフォルダの中身を見てみよう。getdata.htmというファイルができているので，それをブラウザで開けば，Yahoo！のトップページHTMLが表示されるはずである。

なお，LAN環境であればほかに画面に特定の変化はないが，たとえばWindows95環境でダイヤルアップネットワーク環境を設定していると，デフォルトで使用する接続先へアクセスするためのダイアログが表示される。

リスト6　Urlget.java

```
//
// URLのテキストを取得するサンプルJavaアプリケーション Urlget.class
//

import     java.io.*;
import     java.net.*;

public class Urlget {
     public static void main(String args[]) {
          if (args.length == 0){
               System.out.println("[Usage] java Sample3 URLorIPaddress");
          }
          else {
               try {
                    int data;
                    // URLソケットオブジェクト生成
                    URL urlObj;
                    urlObj = new URL(args[0]);
                    DataInputStream inStream;
                    // OpenStream()はurlへ接続し、InputStreamを返す
                    inStream = new DataInputStream(urlObj.openStream());
                    FileOutputStream outStream;
                    outStream = new FileOutputStream("getdata.htm");
                    // 符号なしデータを読み書き
                    for (;;){
                         outStream.write(inStream.readUnsignedByte());
                    }
               }
               catch(EOFException e){
                    // ファイルがEOFに達した場合
                    System.out.println("url downloaded.");
               }
               catch(IOException e){
                    // I/Oエラーが発生した場合
                    System.out.println("I/O Error.");
               }
               catch(Exception e){
                    // それ以外のエラーが発生した場合
                    System.out.println("Error....");
               }
          }
     }
}
```

図3　ストリームの応用法概念図

基本型変数
配列変数
Javaプログラム
入力ストリームのスーパークラス
入力ストリームのサブクラス
ファイル
ディスプレイ
キーボード
ソースとなるデバイス

※スーパークラスで目的とする用途にたえない場合，サブクラスを繋ぎ，そのメソッドを使用して処理を行う。これは入力の例だが，出力でも原理は一緒である。なお，繋げるクラスは標準入力のSystem.inやソケットのクラスなど，他のストリームクラスでもかまわない。

ストリームの応用方法

実例を見たところで，サブクラスを用いた応用の仕方も解説しておく。

とある基本クラスで目的の用途にたえなかった場合，Javaではほかのサブクラスなどを繋ぎ，サブクラス側のメソッドで処理を行うという技法を用いることが可能である。これを概念化したものが図3になる。

では実際に状況を想定し，それに基づいて記述例を挙げてみよう。

まず，特定のファイルからバイナリのデータを取得するJavaアプリケーションを作る必要があるとする。内部フォーマットはすでにわかっているものと仮定しておく。

で，実際に読み込むには先ほどのFileInputStreamでもこと足りるのだが，通常こういったファイルの場合，int型，long型，およびbyte型などが混在しているのが普通で，byte単位の読み込みでは非効率かつ読み込んだあとのデータ整形も少々面倒そうである。もし，このクラスで読み込んだ場合，byte型配列のバッファを用意して，各基本型へビットシフトしながら整形，などというルーチンが別途必要になるだろう。ただでさえ処理が遅いJavaなので，データが大きかったりすると実用にたえないということにもなりかねない。そこで，この仕様の用途にたえうる関数をfile.io.*の中から探すことにしよう。

すると，一覧の中にDataInputStreamというクラスが見つかると思う。このクラスはFileInputStreamのサブクラスで，Javaの持っている基本データ型(int，longなど)の単位で読み込む機能を持っている。このクラスなら今回例の仕様にあっているので，こちらを使うことにする。この場合のコンストラクタだが，以下のように記述すればいい。

```
DataInputStream fileIn =newDataInputStream
(new FileInputStream("ファイル名"));
```

宣言方法に注意してほしい。DataInputStreamコンストラクタの引数として，FileInputStreamのオブジェクトを渡している。処理はDataInputStreamのオブジェクトであるfileIn側のメソッドを使用し，以下のようにファイルから基本データ型を読み込む。

```
int getint = fileIn.readInt();
long getlong = fileIn.readLong();
```

readInt()メソッドはintサイズ(32ビット)のデータを，readLong()メソッドはlongサイズ(16ビット)のデータを返す関数である。ちなみにクローズ処理や例外処理は通常どおりでよい。この例は入力ストリームだが，出力の場合でも基本は一緒である。

もうひとつの例として，標準入力から日本語を得る場合の宣言方法も挙げておこう。この場合，標準入力であるSystem.inを文字列入力ストリームInputStreamReaderに繋げばよい。宣言の例は以下のとおり。

```
InputStreamReader sysIn = new InputStreamReader(System.in);
```

先ほどの例同様，InputStreamReader側のread()メソッドを使用して実際に文字列を読み込むことになる。ここで，System.inにコンストラクタが必要では？　と思った方もおられるだろう。Systemプロパティで扱ったが，これは「finalクラス」という特殊なクラスで，起動時にすでに生成されており，ユーザーがインスタンス生成することもできないというものである。ここで挙げた例のような場合には，いきなり記述してもエラーにはならない。

また，この例のように，繋げるストリームクラスは必ずしもそのクラスのサブクラスとは限らない(なにが使えるのかは，各クラスのコンストラクタの内容を各自確認していただきたい)。なお，I/Oの例で取り上げたBufferedクラスの使い方もこれと同様である。

URLチェックサンプル

図4をご覧いただきたい。これは，登録したサイトのHTMLファイルサイズをチェックして，更新状況をチェックするという，フリーウェアなどでよく見かけるツールのJavaアプリケーション版サンプル，WebCheck.classの実行画面である。

ネットワークの例で挙げたUrlget.classの発展型になっており，URLで指定したサイトのファイルサイズをチェックする。"[]"内の数字はサイトのファイルサイズであり，前回と違う場合はアスタリスクをつけて更新されたことを示してくれる。

もし，サイトが存在しないか，例外が発生した場合はサイズが－1になる。日付を取得できないので完璧ではないが，更新履歴のページなどを設定すれば，ある程度までチェック可能だろう。実行するにはコンソールから以下のように打ち込めばよい。

```
>java WebCheck
```

このサンプルには，いままで解説したURLクラスとBufferedストリームクラスを使用している。画面はFrameにしてあるため，使い勝手を考えて，今回解説しなかったGUIコンポーネントのawtパッケージをあえて使用している。誌面の都合ですべてのソースをここには載せられないが，付属のCD-ROMに添付してあるのでそちらを参照いただきたい。なお，Visual Cafeで開発してあるので，読者の皆さんが改造される場合は，添付の体験版を使用されるといいだろう。

学習用サイトとJavaの今後

これまで解説した機能の情報に詳しいサイトを表4にリストアップした。開発時の参考にしていただきたい。

Javaはまだまだ発展途上であり，解説したこともいずれ仕様変更になってしまう可能性もあるが，勉強しておいて決して損のない技術である。本格的にビジネスで使われるのはおそらくもう少し先になるだろうが。

よくいわれるJavaの遅さも，Just-In-Time(JIT)コンパイラの搭載や，HotSpot技術により，解決は時間の問題といっても過言ではない。実際，Symantec製のJITコンパイラを搭載したJDK1.1.6では，平均して4倍も速度アップしている(ちなみに筆者の空回しループ実験では7倍近い差が出た)。さらに，次バージョンであるJDK1.2では，新機能が多数追加されている。機会があればJDK1.2の機能で作成したツールなどを紹介したいと思う。

この記事で読者の皆さんがJavaに興味を持っていただければ幸いである。

本記事を執筆するうえで，Intercom@西村氏には大変参考になる資料をいただいた。この場を借りてお礼を述べさせていただく。

図4　WebCheck.class

Webページの更新をチェックするツール

表4　参考になるサイト一覧

■Java学習用資料

サイト	URL
Java FAQ 日本語版	http://tech.webcity.ne.jp/˜andoh/java/javafaq.html
JavaDeveloperConnection(JDC)	http://developer.javasoft.com/servlet/SessionServlet?action=showLogin&url=/developer/index.html
Java House ML Topics	http://java-house.etl.go.jp/ml/topics/
JDK1.1.1 日本語版 Top Page	http://java.sun.com/products/jdk/1.1.1/ja/index.ja.html
JDK1.1.1 日本語 Online Document	http://www.agusa.nuie.nagoya-u.ac.jp/person/hachisu/Java/jdk1.1.1/docs/ja/
JDK1.1.1 Document-jar	http://www.agusa.nuie.nagoya-u.ac.jp/person/hachisu/Java/jdk1.1.1/docs/ja/guide/jar/index.html
JDK1.1.1 Document-署名アプレット	http://www.agusa.nuie.nagoya-u.ac.jp/person/hachisu/Java/jdk1.1.1/docs/ja/guide/security/index.html
日経Javaレビュー	http://www2.nikkeibp.co.jp/Java/
Javaカンファレンス	http://www.java-fj.or.jp/

■Javaリンク集

サイト	URL
Digital Cats	http://www.javacats.com/JP/

■Javaメーラー［NOVITA MAIL］

サイト	URL
Novita Communications, Inc.	http://www.novita.com/
Intercom,Inc.(日本語版販売元)	http://www.intercom.co.jp/

マシン語ってなぁに？

影山裕昭/Kageyama Hiroaki

この時代にマシン語？　しかもx86系の？　という声が聞こえてきそうですが，MMXや3D Now！，KNIを使うには必須となります。なにより，システムの深淵に迫るためには避けられない道です。「386以降に限れば，Z80よりは使いやすい」という話ではあるのですが……。とにかく，なにをするにも最終的に頼れるのはアセンブラだけです。アセンブラを使うためにC++が必要になるというのも歪んだ状況を象徴している感じですが……。

現在，アプリケーションプログラムはC，C++言語で開発されるのが主流になりましたが，いまも昔もコンピュータが理解できる唯一の言語はマシン語です。主記憶64Kバイト，8ビットCPUとBASICが全盛だった頃，マシン語プログラマは，ドラゴンをも倒すことのできる戦士のようにいわれていた時代がありました。ハードウェアを直接制御することによってPC本来の性能を引き出し，インタプリタ言語のBASICでは実現不可能な，高速処理のプログラムを開発することができたからです。それはとても凄いことだったのです。その後，CPUのスピードアップとコンパイラの性能向上などを背景に，徐々に開発環境がC言語に移行し，いまやアセンブラ文化は消えつつあります。しかし3次元処理や画像加工処理など，処理速度が要求される部分では，現在でもマシン語が使われています。C，C++言語で実現困難な処理速度も，マシン語であればそれが可能になるからです。

■開発環境を用意する

マシン語（＝アセンブリ言語。コラム参照）でプログラムを書くために絶対必要なツールとして，アセンブラがあります。最初に触れたように，Windows95/98上で動作する多くのアプリケーションプログラムがVisualBasic，Visual C++などの高級言語で開発されています。はたしてWindows95/98上でのアセンブリ言語によるソフトウェア開発の手法を解説した書籍はあるのでしょうか？　少なくとも私は見たことがありません。このような状況のなかで，アセンブリ言語での開発環境を提供するツールを探すのは困難を極めます。

まだMS-DOSが主流だった頃は，アセンブリ言語によるプログラミングもそこそこ行われていました。そんな時代に主流となったアセンブラがMASM（Microsoft Macro Assembler）です。MASMは「マイクロソフトマクロアセンブラプロフェッショナルパッケージ」という商品名で注文すれば現在でも入手は可能です。最新版MASMはVer.6.0になっています。

Windows95/98アプリケーションを開発することを考えれば，MASMは決してよい選択とはいえないかもしれません。なぜなら，Windows95/98のようなマルチタスクOSで動作するアプリケーションをすべてアセンブリ言語で記述することは容易でないからです。

ところで，Visual C++にはインラインアセンブラが用意されています。インラインアセンブラの機能を使えば，C言語のコードの一部にアセンブリ言語を記述することができます。これを利用すればウィンドウ作成に関わる面倒な処理はC++言語で記述し，処理速度を稼ぎたい部分にアセンブリ言語を使うことができます。無論Visual C++なら入手も容易です。

以上の理由から，本格的にアセンブラを勉強する方やMS-DOSからアセンブリ言語でプログラムを開発される予定のある方にはMASM，これからプログラミングの勉強を始めようという方や，C（C++）言語の処理速度に不満を感じている方にはVisual C++でのインラインアセンブラを開発環境に加えてみましょう。ここではVisual C++を開発環境として利用し，インラインアセンブラを使ってみることにします。

■データ型とレジスタ構成

さっそくですが，アセンブリ言語で扱う主なデータ型について説明します。データ型はバイト，ワード，ダブルワード，クワッドワードの4種類に大別できます。

●バイト

1バイトは8ビットで各ビットには0～7までの番号がつけられています。ビット0が最下位ビット（LSB）になっています。

●ワード

1ワードは連続した2バイトでビットには0～15の番号がつけられています。ビット0～7を下位バイト，ビット8～15を上位バイトと呼びます。下位バイトが小さいほうのアドレスの1バイトに格納されます。

マシン語とアセンブリ言語

COLUMN

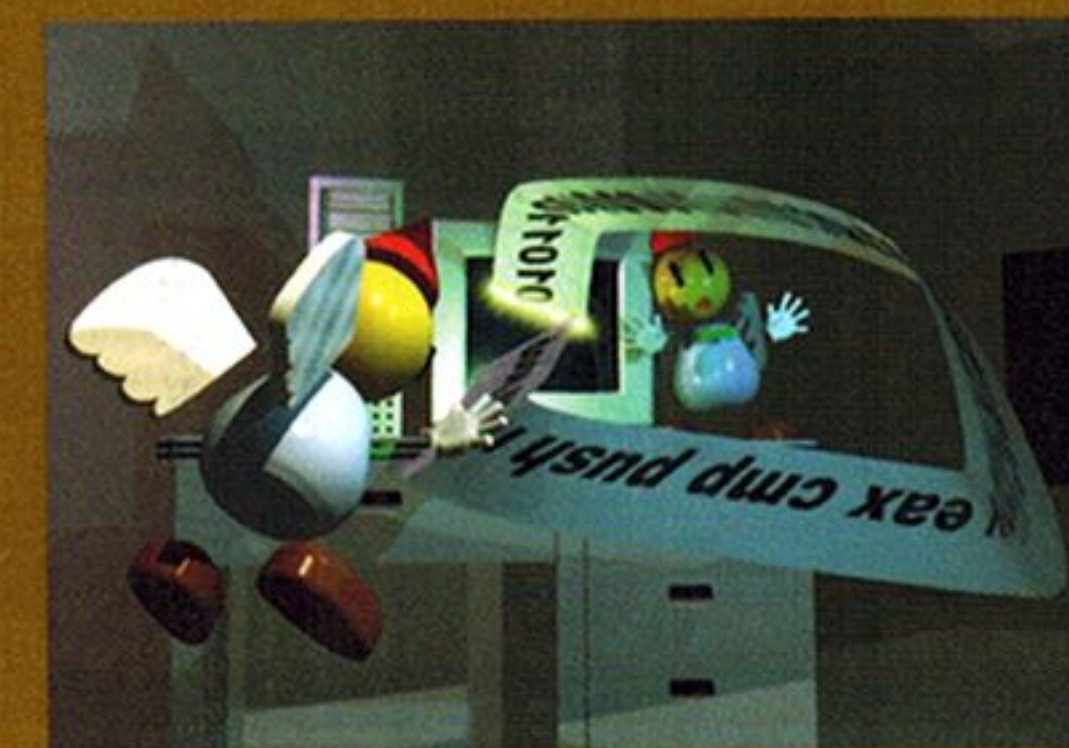
マシン語は0と1の組み合わせからなる2進数で表されます。マシン語は1バイトから数バイト単位で1命令を構成しますが，2進数の羅列を見て，それがどんな命令であるか理解できる人は特殊な方でしょう。0と1の羅列では発音しにくいので，一般には2進数を16進数に変換して表現します。10100011ならa3です。読みやすくはなりましたが，よく分からない数字であることに変わりはありません。まるで暗号のようです。そこで，もっとマシン語を人間にわかりやすいようにしようじゃないか，と考えた出されたのがアセンブリ言語です。マシン語で，

```
b8 00 00 00 00
```

は，アセンブリ言語では，

```
mov eax, 0
```

となります。movはMOVeが由来です。英語が語源であるため，母国語が日本語の私たちには多少のとっつきにくさはありますが，16進数で命令を覚えることに比較すれば，遥かに理解しやすいことがおわかりいただけたと思います。

今度は人間の考えた出したアセンブリ言語をコンピュータが理解できる2進数に変換するツールが必要になります。そのためのツールがアセンブラです。一般にアセンブリ言語とアセンブラは同義語として扱われるようです。「私はアセンブラが書けます」といえば，アセンブリ言語でプログラムを書けるということです。

●ダブルワード

1ダブルワードは連続した4バイトでビットには0～31の番号が付けられています。ビット0～15を下位ワード，16～31を上位ワードと呼びます。下位ワードが小さいほうのアドレスの2バイトに格納されます。

●クワッドワード

1クワッドワードは連続した8バイトでビットには0～63の番号がついています。ビット番号0～31を下位ダブルワード，ビット32～63を上位ダブルワードと呼びます。下位ダブルワードが小さいほうのアドレスの4バイトに格納されます。

●汎用レジスタ

Pentiumプロセッサには8つの汎用レジスタがあります。

```
eax, ebx, ecx, edx,
ebp, esp, esi, edi
```

の8つです。

これらは8086の16ビットレジスタax，bx，cx，dx，bp，sp，si，diレジスタを拡張（Extend）した32ビットレジスタです。32ビットレジスタは16ビットまたは8ビットレジスタに分解することができます。たとえば，eaxの下位ワードはaxと表記します。さらにaxの上位バイトはah，axの下位バイトはalと表記します。

```
mov eax, 0x12345678
```

を実行すると，axには0x5678，ahには0x56，alには0x78が入ります。

アセンブリ言語でプログラミングする場合，変数は汎用レジスタまたはメモリに格納します。どちらでもよい場合は汎用レジスタを使ったほうがいいでしょう。なぜならメモリは動作速度が遅いからです。2次キャッシュは「高速なCPU」と「低速なメモリ」の動作速度のギャップを埋めるために両者の間に存在します。2次キャッシュの動作速度は，内部クロック，外部クロックが同じでも使用するCPUにより変化します。Classic Pentium，MMX Pentiumは，外部クロック＝2次キャッシュの動作速度です。Pentium Proでは内部クロック＝2次キャッシュの動作速度です。さらにPentium IIでは内部クロック÷2＝2次キャッシュの動作速度です。

CPUの違いによるデータ転送能力を検証してみると，233MHzのときMMX Pentiumでは66MHz×64ビット=528Mバイト/秒，Pentium IIでは116.6×64ビット=932.8Mバイト/秒となります。2次キャッシュにミスヒットしたときはメモリにアクセスします。

●セグメントレジスタ

cs，ds，ss，es，fs，gsという16ビットレジスタです。これらはメモリを参照する際に使われます。Visual C++からインラインアセンブラを使う分には，利用することはあまりないでしょう。

●フラグレジスタ

算術演算，比較演算，ビット操作などを行うと，その演算結果がフラグレジスタ（表1）に反映されます。

フラグレジスタは主に条件分岐に使用されます。ループ処理などで多く使われるのは，ZFフラグ（ゼロフラグ）です。ZFフラグは演算結果がゼロになったときに1，それ以外は0になります。フラグが1になることを「フラグをセットする」といいます。フラグが0になることは「フラグをクリアする」ということが多いようです。

たとえば，eaxレジスタにループカウンタをセットしたとすると，

```
loop: dec eax   ; eax--;
      jnz loop  ; if (eax!=0) goto loop;
```

となります。jnz loopはジャンプ・ノン・ゼロ（jump non zero）の意味で，ゼロでなかったら続くアドレス（ここではloop）にジャンプせよ，という命令です。

次に大小比較を行った結果，3種類の処理に分岐する例を示します。

```
cmp eax, ebx
je  equal   ; if (eax==ebx) goto equal;
jb  small   ; if (eax<ebx) goto small
jl  large   ; if (eax>ebx) goto large
```

cmp eax，ebxはsrcオペランドとdestオペランドの比較（CoMPare）を行う命令です。この例ではeax－ebxの演算結果がフラグレジスタに格納されます。eax，ebxとも演算結果によるレジスタ内容の変更はありません。

またフラグレジスタにはディレクションフラグというものがあります。これはストリング命令を使った際の動作を決定するものです。詳しくはストリング命令の説明の項で行います。

■実効アドレス計算

実効アドレスは表2の計算式より求められます。

図1

表1　フラグレジスタ

名前	目的	状態
OF	オーバーフロー	結果が正または負の有効範囲を超える
SF	符　号	結果が負（ゼロより小さい）
ZF	ゼ　ロ	結果がゼロ
PF	パリティ	結果の下位バイトに偶数パリティがある
CF	キャリー	結果の最下位ビットからキャリーアウトした

表2　実効アドレス計算

ベースレジスタ＋（インデックスレジスタ×スケールファクタ）＋ディスプレースメント

・ベースレジスタはeax，ebx，ecx，edx，ebp，esp，esi，ediからひとつを選択
・インデックスレジスタはeax，ebx，ecx，edx，ebp，esp，esi，ediからひとつを選択
・スケールファクタは1，2，4，8からひとつを選択
・ディスプレースメントは8ビットまたは32ビットの符号付き整数（省略化）

たとえばint buffer[100]の配列要素ebxにアクセスするには,

```
mov   eax, dword ptr [buffer+ebx※4]
```

と書くことができます。これはeax=buffer[ebx]と同じ意味になります。[]内で4倍しているのは,integer配列の1要素の大きさが4バイトだからです。

■アプリケーションプログラミング

これよりしばらくはPentiumプロセッサ用の整数命令の概要を説明します。BCD命令や割り込み制御など,利用頻度の少ないと思われる命令の紹介は省略しています。さらにFPUに対して浮動小数点オペランドを操作する命令や,その他MMX命令などがありますが,これらについては別の機会で取り上げたいと思います。

●データ移動命令

mov(Move)

メモリ⇔レジスタ間転送,汎用レジスタ間転送,レジスタ,メモリに対する即値データ転送に使います。メモリ間のデータ転送にはmovs命令を使います。

xchg(Exchange)

異なるレジスタ間,またはメモリとレジスタの内容を交換する命令です。オペランドサイズはバイト,ワード,ダブルワードが指定できます。当然2つのオペランドサイズは同じにしなければいけません。

●スタック操作命令

push(Push)/**pop**(Pop)

push命令はスタックポインタをデクリメントしたあと,スタックポインタの最上段にソースオペランドの内容をコピーする命令です。デクリメントする数はソースオペランドのサイズにより決定します。バイトのときは1,ワードのときは2,ダブルワードのときは4,デクリメントします。ソースオペランドには,メモリアドレス,即値,レジスタを指定することができます。push命令は一時的にレジスタの値を保管しておきたい場合などに使用されます。

pop命令はpush命令の逆の動作をします。つまりスタックポインタの最上段にある内容をデスティネーションオペランドにコピーし,スタックポインタをインクリメントします。インクリメントする数は,push命令のときと同じ規則で決まります。push命令で退避した内容を戻す場合に使われます。

pusha(Push All Registers)/
popa(Pop All Registers)

pusha命令は8つの汎用レジスタの内容をスタックにコピーする命令です。コピーする順番はeax,ecx,edx,ebx,eaxをpushする前のesp,ebp,esi,ediとなっています。

popa命令はpusha命令でスタックにコピーした内容をレジスタに戻す命令です。pusha/popa命令はpush/pop命令に比較して実行クロックが余計にかかるので,あまり使われることのない命令でしょう。

●型変換命令

cwd(Convert Word to Doubleword)/
cdq(Convert Doubleword to Quad-Word)

cwd命令はaxレジスタの符号(ビット15)をdxレジスタの各ビットにコピーします。cdq命令はeaxレジスタの符号ビット(ビット31)をedxレジスタの各ビットにコピーします。これらは除算に使う被除数を設定するために使われます。

cbw(Convert Byte to Word)

alレジスタの符号(ビット7)をahレジスタの各ビットにコピーする命令です。

cwde(Convert Word to Doubleword Extend)

axレジスタの符号(ビット15)をeaxレジスタの上位ワードの各ビットにコピーする命令です。

movsx(Move with Sign Extension)

movsx命令はメモリやレジスタの内容を1バイトまたは1ワード読み込み,命令のオペランドサイズ属性(1ワードまたは1ロングワード)に符号ビットを拡張して,デスティネーションレジスタに格納します。

Movzx(Move with Zero Extension)

movzx命令はメモリやレジスタの内容を1バイトまたは1ワード読み込み,命令のオペランドサイズ属性(1ワードまたは1ロングワード)にゼロ拡張して,デスティネーションレジスタに格納します。

●加算命令・減算命令

add(Add Integers)

ソースオペランドとデスティネーションオペランドの合計をデスティネーションオペランドに格納します。

adc(Add integers with Carry)

ソースオペランドとデスティネーションオペランドの合計にキャリーフラグの値を加算したものをデスティネーションオペランドに格納します。add命令の代わりにadc命令を使うことによって桁上がりを次の計算に含めることができます。たとえば32ビットレジスタを組み合わせてクワッドワードの加算を行う場合などに便利です。

inc(Increment)

デスティネーションオペランドに1加算します。inc命令はCFフラグに影響を与えません。

sub(Subtract Integers)

デスティネーションオペランドからソースオペランドを減算した結果をデスティネーションオペランドに格納します。

sbb(Subtract Integers with Borrow)

デスティネーションオペランドからソースオペランドを減算した結果からCFフラグの値を減算したものをデスティネーションオペランドに格納

Push/Popの動作

します。桁下がりが必要な場合はCFフラグが立ちます。sub命令の代わりにsbb命令を使うことによって，桁下がりを次の減算に含めることができます。たとえば32ビットレジスタを組み合わせてクワッドワードの減算をしたい場合などに便利です。

dec(Decrement)

デスティネーションオペランドから1減算します。dec命令はCFフラグに影響を与えません。

●比較命令と符号変更命令

cmp(Compare)

デスティネーションオペランドからソースオペランドを減算します。計算結果はデスティネーションオペランドに格納されずに，OF，SF，ZF，AF，PF，CFのフラグのみ更新します。計算結果は条件付き命令によってテストすることができます。

neg(Negate)

ゼロから符号付き整数オペランドを減算します。計算結果は符号を逆転したものになります。

●乗算命令

mul(Unsigned Integer Multiply)

ソースオペランドと，al，ax，eaxの各レジスタと符号なしの乗算を実行します。ソースオペランドがバイトの場合，alレジスタと乗算を行い，結果を倍長のaxレジスタに格納します。ソースオペランドがワードの場合はaxレジスタと乗算を行い，結果をeaxレジスタに格納します。最後にソースオペランドがダブルワードの場合は，eaxレジスタと乗算を行い，結果をedxとeaxのクワッドワードに格納します。結果の値の上半分がゼロ以外であれば，CFとOFの各フラグがセットされます。ゼロのときは各フラグがクリアされます。

imul(Signed Integer Multiply)

符号付き乗算を実行します。この命令は3つのオペランド形式があります。

1）1オペランド形式
ソースオペランドにはメモリ，または汎用レジスタに格納されたバイト，ワード，ダブルワードのいずれかを指定します。ソースオペランドとal，ax，eaxの各レジスタを乗算します。結果をソースオペランドがバイトのときはax，ワードのときはeax，ダブルワードのときはedxとeaxに格納します。

2）2オペランド形式
一方のソースオペランド（第1オペランド）に汎用レジスタ，もう一方（第2オペランド）に汎用レジスタかメモリを指定します。第1オペランドの汎用レジスタに第1オペランド×第2オペランドの計算結果が格納されます。

3）3オペランド形式
デスティネーションオペランド（第1オペランド）に汎用レジスタ，ソースオペランド（第2オペランド）に汎用レジスタまたはメモリ，ソースオペランド（第3オペランド）に即値を指定します。結果はデスティネーションオペランドに格納されます。

いずれの場合も有意ビットが結果の上半分にキャリーされる場合，CF，OFフラグがセットされます。また結果の上半分が下半分の符号拡張部分に相当する場合は，CF，OFフラグはクリアされます。

●除算命令

div(Unsigned Integer Divide)

ソースオペランドの値により，al，ax，eaxのレジスタについて符号なしの除算を実行します。被除数，除数，商，剰余の関係は表を参照してください。計算結果が整数でなかった場合，小数点以下は切り捨てられます。剰余が商より大きいことはありません。

idiv(Signed Integer Divide)

div命令を符号付きで実行する命令です。レジスタの使用規則もdiv命令と同じです。

●論理演算命令

not(Not)

ソースオペランドの各ビットを反転させます。オペランドにはレジスタ，メモリを指定します。

and，or，xor

論理積，論理和，排他的論理和の標準論理演算を行います。次のオペランドを組み合わせて使用できます。

- 2つのレジスタオペランド
- メモリオペランドを含む汎用レジスタオペランド
- 汎用レジスタオペランド，またはメモリオペランドを含む即値オペランド

実行後，OF，CFフラグがクリアされ，SF，ZF，PFフラグが更新されます。

```
and eax, 0xffff0000
```

はeaxレジスタの下位ワードをクリアします。上位ワードがゼロの場合，ZFフラグがセットされます。

●ビットテスト/修正命令

bt(BitTest)

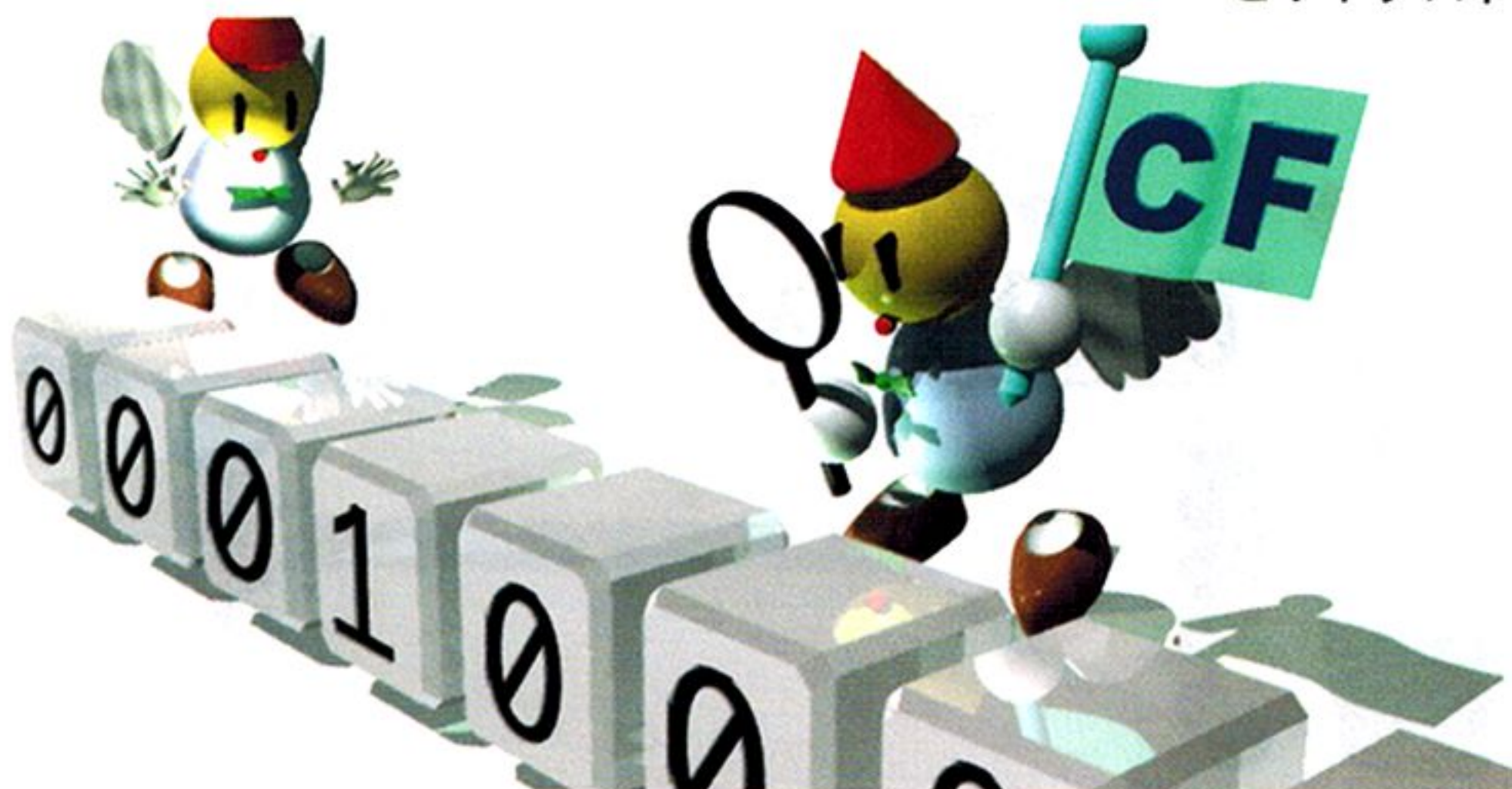
メモリ，汎用レジスタの特定ビットを対象に操作する命令です。オペランドの最下位ビットからのオフセット値を指定します。オフセット値は汎用レジスタ，即値バイトで指定します。たとえば，

```
bt ax, 5
```

はaxレジスタの第5ビットをテストします。第5

表3

オペランドサイズ(除数)	被除数	商	剰余
バイト	axレジスタ	alレジスタ	ahレジスタ
ワード	dxおよびaxレジスタ	axレジスタ	dxレジスタ
ダブルワード	edxおよびeaxレジスタ	eaxレジスタ	edxレジスタ

ビットテスト

ビットが1のとき，CFフラグはセットされます。それ以外はCFフラグはクリアされます。

●ビットスキャン命令

bsf(Bit Scan Forward)
bsr(Bit Scan Reverse)

ビットスキャン命令はワードあるいはダブルワードの第2オペランド中のビットを，ビット位置0から上位方向に向かって走査します。全ビットがゼロのとき，ZFフラグをセットします。そうでなければ最初に見つけたゼロ以外のビット番号をデスティネーションレジスタに格納します。bsr命令は最上位ビットから最下位ビットに向けて走査します。

```
mov eax, 4
bsf ebx, eax
```

ebxレジスタには2が格納され，ZF＝0にクリアされます。

```
xor eax, eax
bsr ebx, eax
```

ZF=1にセットされます。

●シフト/ローテート命令

sal(Shift Arthmetic Left)

デスティネーションのバイト，ワード，ダブルワードオペランドを1ビット，またはカウントオペランド(clレジスタか即値)で指定したビット数だけ左にシフトする命令です。シフト後，最上位ビットの値がCFフラグに格納されます。空ビット位置はクリアされます。sal命令とまったく同じ動作をするshl(Shift Logical Left)という命令がありますが，これはアセンブラが便宜上用意している命令です。

shr(Shift Logical Right)

デスティネーションのバイト，ワード，ダブルワードオペランドを1ビット，またはカウントオペランド(clレジスタか即値)で指定したビット数だけ右にシフトする命令です。

シフト後，最下位ビットの値がCFフラグに格納されます。空ビット位置はクリアされます。

sar(Shift Arthmetic Right)

デスティネーションのバイト，ワード，ダブルワードオペランドを1ビット，またはカウントオペランド(clレジスタか即値)で指定したビット数だけ右にシフトする命令です。シフト後，最下位ビットの値がCFフラグに格納されます。最上位ビット(符号ビット)の値は変更されず，空ビット位置には最上位ビットの値がコピーされるため，シフト後も符号情報を保持しています。

●ダブルシフト命令

shld(Shift Left Double)

デスティネーションのワード，ロングワードを左にシフトして，空ビット位置にソースオペランドからシフトされたビットを格納します。いちばん最後に最上位ビットからシフトされた値がCFフラグに格納されます。シフトするビット数はclレジスタか即値で指定します。ソースオペランドは変更されません。

shrd(Shift Right Double)

デスティネーションのワード，ロングワードを右にシフトして，空ビット位置にソースオペランドからシフトされたビットを格納します。いちばん最後に最下位ビットからシフトされた値がCFフラグに格納されます。シフトするビット数はclレジスタか即値で指定します。ソースオペランドは変更されません。

●ローテート命令

rol(Rotate Left)

デスティネーションのバイト，ワード，ダブルワードオペランドを1ビット，またはカウントオペランド(clレジスタか即値)で指定したビット数だけ左にシフトする命令です。シフト後，最上位ビットの値が最下位ビットとCFフラグに格納されます。

ror(Rotate Right)

デスティネーションのバイト，ワード，ダブルワードオペランドを1ビット，またはカウントオペランド(clレジスタか即値)で指定したビット数だけ右にシフトする命令です。シフト後，最下位ビットの値が最上位ビットとCFフラグに格納されます。

rcl(Rotate Through Carry Left)

デスティネーションのバイト，ワード，ダブルワードオペランドを1ビット，またはカウントオペランド(clレジスタか即値)で指定したビット数だけ左にシフトする命令です。rol命令と違うのはCFフラグをデスティネーションオペランドの最上位ビットとして扱うことです。

rcr(Rotate Through Carry Right)

デスティネーションのバイト，ワード，ダブルワードオペランドを1ビット，またはカウントオペランド(clレジスタか即値)で指定したビット数だけ右にシフトする命令です。ror命令と違うのはCFフラグをデスティネーションオペランドの最下位ビットとして扱うことです。

●条件付きバイトセット命令

setcc(Set Byte on Condition cc)

フラグレジスタの状態により動作を決定します。条件ccが真の場合はバイトに1をセットします。条件を満たさない場合はバイトをクリアします。

表4

命令	CFフラグに対する影響	選択されたビットに対する影響
bt(Bit Test)	CF←選択されたビット	影響なし
bts(Bit Test and Set)	CF←選択されたビット	選択されたビット←1
btr(Bit Test and Reset)	CF←選択されたビット	選択されたビット←0
btc(Bit Test and Complement)	CF←選択されたビット	選択されたビット←選択されたビットの補数

ローテート

●テスト命令

test(Test)

2つのオペランドの論理積(AND)を取って，OF，CFフラグをクリアします。SF，ZF，PFの各フラグが更新されます。フラグのみの変更で，デスティネーションオペランドは変更されません。結果は条件付き制御転送命令，条件付きバイトセット命令で取得することができます。

●ジャンプ命令

jmp(Jump)

無条件に別のルーチンに実行制御を渡します。相対アドレスまたは絶対アドレス指定がありますが，アセンブラが自動的に解釈してくれます。

●コール命令

call(Call)

無条件に実行制御を別のルーチンに渡しますが，call命令の次の命令が格納されているアドレスをスタックに格納します。制御を移したルーチンで後述のret命令を実行することにより，call命令の次のアドレスに制御を戻すことができます。

●リターン命令

ret(Return From Procedure)

call命令とペアで使用します。実行制御をcall命令の次の命令に渡します。

●条件付きジャンプ命令

指定した条件が真の場合に，実行制御を別のルーチンに渡します。(表5)

●ループ命令

これらの命令は暗黙的にecxレジスタをカウントレジスタとして使用します。loop命令を使う場合，ecxレジスタ以外はカウントレジスタとして使用することができません。ほかのレジスタを使用したい場合は，dec命令とjnz命令を使用します。

loop(Loop While ECX Not Zero)

ecxレジスタで指定した回数だけ指定アドレスへジャンプする命令です。

```
            xor  eax, eax   ; int eax=0;
            mov  ecx, 10    ; int ecx=10;
kurikaesi:  add  eax, ecx   ; eax+=ecx;
            loop kurikaesi  ; ecx--
                            ; if (ecx > 0) goto kurikaesi;
```

このプログラムはeaxレジスタに1～10の和を求めるものです。loop命令は最初にecxレジスタを1減算します。その結果ecxレジスタがゼロでなければ，オペランドの指定アドレスへ制御を渡します。loop命令の飛び先アドレスは－128バイト～＋127バイトの制限があります。

loope(Loop While Equal)/loopz(Loop While Zero)

ecxレジスタで指定した回数だけ指定アドレスへジャンプしますが，ZFフラグがクリアされた時点でloope/loopz命令の直後の命令に実行制御を渡します。

loopne(Loop While Not Equal)/loopnz(Loop While Not Zero)

ecxレジスタで指定した回数だけ指定アドレスへジャンプしますが，ZFフラグがセットされた時点でloopne/loopnz命令の直後の命令に実行制御を渡します。

jecxz(Jump if ecx Zero)

ecxレジスタが0になった時点で指定されたデスティネーションにジャンプします。

●ストリング命令

ストリング命令のうちmovs命令は，ソースストリングとデスティネーションストリングを使います。ソースストリングはesiレジスタ，デスティネーションストリングはediレジスタでアドレス指定します。ストリング命令を実行すると，自動的にesi，ediレジスタがインクリメント(加算)，またはデクリメント(減算)されます。加算か減算かはディレクションフラグによって決められます。ディレクションフラグ＝0のときに加算，1のときに減算します。ディレクションフラグを0にするにはcld(Clear Direction Flag)命令，1にするにはstd(Set Direction Flag)命令を使用します。

```
cld
mov  esi, source
mov  edi, dist
movsd
```

上記プログラムはmovsd命令により，esiレジスタで示されるアドレスからediレジスタで示さ

表5

ニーモニック	フラグ状態	説 明
	符号なし条件付きジャンプ	
ja/jnbe	(CF or ZF)=0	より大きい/以下でない
jae/jnb	CF=0	以上/未満でない
jb/jnae	CF=1	より小さい/以上でない
jbe/jna	(CF or ZF)=1	以下/より大きくない
jc	CF=1	キャリー
je/jz	ZF=1	等しい/ゼロ
jnc	CF=0	キャリーなし
jne/jnz	ZF=0	不等/ゼロでない
jnp/jpo	PF=0	パリティなし/奇数パリティ
jp/jpe	PF=1	パリティ/偶数パリティ
	符号付き条件付きジャンプ	
jg/jnle	((SF xor OF) or ZF)=0	より大きい/以下でない
jge/jnl	(SF xor OF)=0	以上/未満でない
jl/jnge	(SF xor OF)=1	より小さい/以上でない
jle/jng	((SF xor OF) or ZF)=1	以下/より大きくない
jno	OF=0	オーバーフローなし
jns	SF=0	符号なし（負数でない）
jo	OF=1	オーバーフロー
js	SF=1	符号あり（負数）

表6

ストリング命令	動作	影響フラグ
movs/movsb/movsw/movsd	[esi]→[edi] esi+4→esi edi+4→edi	なし
cmps/cmpsb/cmpsw/cmpsd	[esi]-[edi] esi+4→esi edi+4→edi	OF，SF，ZF AF，PF，CF
scas/scasb/scasw/scasd	eax-[edi] edi+4→edi	OF，SF，ZF AF，PF，CF
lods/lodsb/lodsw/lodsd	[esi]→eax esi+4→esi	なし
stos/stosb/stosw/stosd	eax→[edi] edi+4→edi	なし

れるアドレスへ4バイトの内容が転送されます。そのあと，ESI，EDIレジスタは自動的に+4加算されます。なおワード転送（movsw）なら+2，バイト転送（movsb）なら+1加算されます。ストリング命令の一覧を表6に示します。

●リピートプリフィックス

ストリング命令にリピートプリフィックスを指定すると，指定の終了条件のひとつが満たされるまで，処理が繰り返し実行されます。

```
cld
mov  esi, source
mov  edi, dist
mov  cx, 100
rep  movsd
```

repプリフィックスはcxレジスタに繰り返し回数を指定します。上記プログラムはsourceで示されるアドレスからdistで示されるアドレスに400バイトブロック転送します。リピートプリフィックス命令の一覧を表7に示します。

●フラグ転送命令

lahf(Load AH from Flags)

SF，ZF，AF，PF，CFの各フラグをahレジスタの7，6，4，2，0ビットにコピーします。

sahf(Load AH from Flags)

lahf命令とは逆に，ahレジスタの7，6，4，2，0の各ビットをSF，ZF，AF，PF，CFフラグにコピーします。

●アドレス計算命令

lea(Load Effective Address)

lea命令はデスティネーションの汎用レジスタにメモリアドレスを設定します。ストリング命令実行前にesi，ediレジスタに，転送元アドレス，転送先アドレスを設定する場合などに使用します。

■2つの実行ユニット

ここまでPentiumプロセッサの命令を長々と紹介してきました。ところで，PentiumプロセッサにはUパイプ，Vパイプと呼ばれる2つの整数実行ユニットがあります。UパイプとVパイプは2つの整数命令を1クロックサイクルで実行できます。このような実行をスーパースカラ実行と呼びます。UパイプはPentiumプロセッサのすべての命令を実行できますが，Vパイプは以下の簡単な命令に限られています。

```
mov  reg, reg/mem/imm
mov  mem, reg/imm
alu  reg, reg/mem/imm
alu  mem, reg/imm
inc  reg/mem
dec  reg/mem
push reg/mem
pop  reg
lea  reg, mem
jmp/call/jcc near
nop
```

UパイプとVパイプで実行される2つの命令が1クロックサイクルで別々に実行されるとペアリングが発生します。あとから実行される命令は直前の命令とペアリングされるために，1クロックサイクル短縮されます。アセンブリ言語でプログラムを書くときは，コードの配置に注意してペアリングを多く作るようにしましょう。

```
1  mov  eax, ebx
2  mov  ecx, edx
```

命令1，2はペアリングされます。

```
1  and  ecx, 0x00ff00ff
2  mov  [eax], ecx
3  and  edx, 0x00ff00ff
4  mov  [ebx], edx
```

命令2のソースオペランドのecxは，命令1で内容が変更されています。例のように連続する2命令で依存関係があるとき，ペアリングはされません。上の例では命令2と3だけがペアリングされます。その結果，3クロックサイクルかかります。

```
1 and  ecx, 0xff00ff
3 and  edx, 0xff00ff
2 mov  [eax], ecx
4 mov  [ebx], edx
```

上記のように書き換えることにより，命令1と3，命令2と4がそれぞれペアリングされます。最適化前に比較して1クロックサイクル短縮されます。

■アセンブラ実践

それではVisual C++のインラインアセンブラを使ってプログラムを組んでみましょう。プログラムの題材としては，最初から画像加工をやろうと心に決めていました。しかし，恥ずかしながら，いままで画像加工処理について勉強したことがありません。なにかヒントを見つけようとOh!Xのバックナンバーを読んでいると，X-BASICにして30行程度のプログラムでモーションブラー効果をかけるものが見つかりました。参考にしたのはOh!X 1992年2月号で中野修一氏の書かれた『画像は加工される』の34ページのリスト2です。元のプログラムは256×256ドットハイカラーの画像に対して，画像中心に向かっての動きを表現するものでした。これを320×240ドットフルカラー画像に対応させました。実行にあたってはDirectDraw対応のビデオカードと，Windows95/98にDirectDrawがインストールされていることが必要です。

通常DirectDrawはプライマリサーフェイス，バックサーフェイス，オフスクリーンサーフェイスと呼ばれる3枚のサーフェイスをフリップ&ブリットして使われますが，今回はプライマリサーフェイスだけ確保し，プライマリサーフェイスを直接アクセスすることにしました。

sample.exeを実行すると，実行ディレクトリにあるsample.bmpを読み込んでモーションブラー効果をかけます。画面左上に元画像，右上にアセンブリ言語でのモーションブラー実行結果，左下にC++言語でのモーションブラー実行結果を描画します。sample.bmpは，24ビットカラーであればほかの画像ファイルをリネームして表示することができます。画像サイズは320×240を前提

表7

リピートプリフィックス	終了条件1	終了条件2
rep	ecx=0	なし
repe/repz	ecx=0	ZF=0
repne/repnz	ecx=0	ZF=1

表8

命令	効果
stc(Set Carry Flag)	CF←1
clc(Clear Carry Flag)	CF←0
cmc(Complement Carry Flag)	CF←-CF
cld(Clear Direction Flag)	DF←0
std(Set Direction Flag)	DF←1

にしていますが，640×480の画像であっても画像の一部を表示することができます。終了するにはESCキーを押してください。判定は左下の画像を表示し終わったあとに行いますので，終了するまでにしばらく時間がかかることがあります。

さて，Visual C++でアセンブリ言語を記述するには，

```
_asm{
  アセンブリ言語
     ・
     ・
}
```

と，_asmで始めて {} の間にアセンブリ言語を記述します。サンプルプログラムではプライマリサーフェイスに対するカラーコードの読み出し，書き込み処理部分をC++のコードで記述しています(GetPoint，DrawPoint)。アセンブリ言語からC++で記述した引数付きのサブルーチンを呼び出すには，いちばん最後の引数から順番にスタックに積んでいきます。たとえば，

```
sample(int a,  int b,  int c)
```

という引数を持つ関数を呼び出すときは，c，b，aの順番でスタックに引数を積んでからcallsampleです。サブルーチンの処理前後で破壊されたくないレジスタがある場合は，push，popを忘れずにしてください。スタックポインタの回復も忘れずに。

■いざ実行！ 結果はいかに？

さて，作成したsample.exeを実行してみると，コンパイラの吐き出したコードとアセンブリ言語でのコードの実行時間にほとんど差がないことに愕然としました。その最大の理由として考えられるのは，アセンブリ言語のコードからC++で書いたサブルーチンを呼び出していること。もうひとつは私の技術力不足です。

実は私自身Z80，68000でのアセンブリ言語でのプログラム経験は多少あるのですが，Pentiumを扱ったのは今回が初めてでした。命令のペアリングなどいろいろ最適化らしきことをやってみたのですが，残念ながら体感できる効果は得られませんでした。単純(というか姑息な手段)ではありますが，ループ展開とか，ある程度座標系のテーブルを用意してあげれば，実行速度はそれなりに速くなるでしょうが，それならC++でもできることですし。アセンブリ言語の魅力を充分に伝えることのできないサンプルプログラムになってしまったことが，かなり残念です。

しかし，現段階の私のPentiumプロセッサに対する知識は，相撲の世界でいえば幕下クラス程度です。冒頭の話にたとえれば，私はドラゴンを倒すことができない戦士なのです。それでもそれなりのプログラムが書けるのですから，もう少し時間をかけて取り組めばもっと速いコードが書ける自信はあります。特にDirectDraw関係の知識がもうちょっとあれば，すべての処理をアセンブリ言語で書けるでしょうしね。また，最適化をするのにも，エクセルソフト株式会社のVTuneを利用すれば，より効率のよい開発環境になるはずです。

今回，アセンブリ言語で書いたソースコードを最適化するために，試行錯誤で細かい調整をしました。コンピュータの奥深い部分を直接操作していると実感できる時間です。このあたりの感覚は高級言語のプログラミングでは味わえない，アセンブリ言語によるプログラミングの醍醐味です。C++で書いたソースコードを最適化するにも，アセンブリ言語の知識は必須のはずです。いままでアセンブリ言語を使ったことのない方も，ぜひ一度アセンブリ言語でのプログラミングに挑戦してみてください。

画面 1

リスト1

```
#include "vgamain.h"

BOOL init(){
    if (LoadBitmap("sample.bmp",320,240) != TRUE){
        return FALSE;
    }
    return TRUE;
}

BOOL main_asm(){
    unsigned int r,g,b;
    unsigned int r1,g1,b1;
    unsigned int r2,g2,b2;
    unsigned int r3,g3,b3;
    unsigned int R =0,G =0,B =0;
    unsigned int R1=0,G1=0,B1=0;
    unsigned int R2=0,G2=0,B2=0;
    unsigned int R3=0,G3=0,B3=0;
    int l[8];
    int i,j,x,y;

    _asm{
                        // k＊k/8CAEeA[EuEaC cIe (k=0A`7)
            mov         ecx,7               // k=7;
maketbl:    mov         eax,ecx             // do {
            mul         ax                  //    eax=k＊k
            shr         eax,3               //    eax=eax/8;
            mov         [ecx＊4+l],eax      //    l[k]=eax
            dec         ecx                 //    k--;
            jns         maketbl             // } while ( k >= 0);

                                            // i=159; j=119; k=7;
            mov         [j],119             // do {
loop_y:     mov         [i],159             //   do {
loop_x:     mov         ecx,7               //     do {
sampling:   mov         ebx,160
            mov         esi,[i]
            sub         ebx,esi             //        ebx=160-i;

            mov         eax,[ecx＊4+l]      //        eax=l[k]
            push        ecx
            imul        ebx                 //        eax=l[k]＊ebx;
            mov         ebx,49
            div         bl                  //        al=eax/49;
            movsx       eax,al
            neg         eax                 //        eax=(int)al＊-1;
            sub         esi,eax             //        esi=i-eax;
            mov         [x],esi             //        x=esi;

            mov         esi,[j]
            mov         ebx,120
            mov         eax,[ecx＊4+l]      //        eax=l[k]
            sub         ebx,esi             //        ebx=120-j;
            imul        ebx                 //        eax=l[k]＊ebx;
            mov         ebx,49
            div         bl                  //        al=eax/49;
            movsx       eax,al
            neg         eax                 //        eax=(int)al＊-1;
            sub         esi,eax             //        esi=j-eax;

            lea         eax,[b]
            lea         ebx,[g]
            lea         ecx,[r]
            mov         [y],esi             //        y=esi;
```

```
		push	eax
		push	ebx
		push	ecx
		mov	eax,[y]
		mov	ebx,[x]
		push	eax
		push	ebx
		call	GetPoint	//  GetPoint(x, y, &r, &g, &b);
//		add	esp,0x14

		lea	eax,[b1]
		lea	ebx,[g1]
		lea	ecx,[r1]
		mov	[esp+16],eax
		mov	eax,239
		mov	[esp+12],ebx
		sub	eax,[y]
		mov	[esp+8],ecx
		mov	[esp+4],eax
		call	GetPoint	// GetPoint(x, 239-y, &r1, &g1, &b1);
//		add	esp,0x14

		lea	eax,[b2]
		lea	ecx,[r2]
		lea	ebx,[g2]
		mov	[esp+16],eax
		mov	[esp+8],ecx
		mov	[esp+12],ebx
		mov	eax,[y]
		mov	ebx,319
		mov	[esp+4],eax
		sub	ebx,[x]
		mov	[esp+0],ebx
		call	GetPoint	// GetPoint(319-x, y, &r2, &g2, &b2);
//		add	esp,0x14

		lea	eax,[b3]
		lea	ecx,[r3]
		lea	ebx,[g3]
		mov	[esp+16],eax
		mov	eax,239
		mov	[esp+12],ebx
		mov	ebx,319
		mov	[esp+8],ecx
		sub	eax,[y]
		mov	[esp+4],eax
		sub	ebx,[x]
		mov	[esp+0],ebx
		call	GetPoint	//GetPoint(319-x, 239-y, &r3, &g3, &b3);
		add	esp,0x14

		mov	eax,[r]
		mov	ebx,[g]
		mov	ecx,[b]
		add	[R],eax		//	R+=r;
		add	[G],ebx		//	G+=g;
		add	[B],ecx		//	B+=b;
		mov	eax,[r1]
		mov	ebx,[g1]
		mov	ecx,[b1]
		add	[R1],eax	//	R1+=r1;
		add	[G1],ebx	//	G1+=g1;
		add	[B1],ecx	//	B1+=b1;
		mov	eax,[r2]
		mov	ebx,[g2]
		mov	ecx,[b2]
		add	[R2],eax	//	R2+=r2;
		add	[G2],ebx	//	B2+=b2;
		add	[B2],ecx	//	G2+=g2;
		mov	eax,[r3]
		mov	ebx,[g3]
		mov	ecx,[b3]
		add	[R3],eax	//	R3+=r3;
		add	[G3],ebx	//	B3+=b3;
		add	[B3],ecx	//	G3+=g3;
		pop	ecx
		dec	ecx		//	k--;
		jns	sampling	//    } while ( k >= 0);

		xor	edx,edx
		mov	eax,[R]
		mov	ecx,[B]
		mov	ebx,[G]
		shr	eax,3
		shr	ebx,3
		shr	ecx,3
		mov	[r],eax		//	r=R/8;
		mov	[g],ebx		//	g=G/8;
		mov	[b],ecx		//	b=B/8;
		mov	[R],edx		//	R=0;
		mov	[G],edx		//	G=0;
		mov	[B],edx		//	B=0;

		mov	eax,[R1]
		mov	ebx,[G1]
		shr	eax,3
		mov	ecx,[B1]
		shr	ebx,3
		shr	ecx,3
		mov	[r1],eax	//	r1=R1/8;
		mov	[g1],ebx	//	g1=G1/8;
		mov	[b1],ecx	//	b1=B1/8;
		mov	[R1],edx	//	R1=0;
		mov	[G1],edx	//	G1=0;
		mov	[B1],edx	//	B1=0;

		mov	eax,[R2]
		mov	ebx,[G2]
		shr	eax,3
		mov	ecx,[B2]
		shr	ebx,3
		shr	ecx,3
		mov	[r2],eax	//	r2=R2/8;
		mov	[g2],ebx	//	g2=G2/8;
		mov	[b2],ecx	//	b2=B2/8;
		mov	[R2],edx	//	R2=0;
		mov	[G2],edx	//	G2=0;
		mov	[B2],edx	//	B2=0;

		mov	eax,[R3]
		mov	ebx,[G3]
		shr	eax,3
		mov	ecx,[B3]
		shr	ebx,3
		shr	ecx,3
		mov	[r3],eax	//	r3=R3/8;
		mov	[g3],ebx	//	r3=G3/8;
```

```
		mov	[b3],ecx	//	b3=B3/8;
		mov	[R3],edx	//	R3=0;
		mov	[G3],edx	//	G3=0;
		mov	[B3],edx	//	B3=0;

		mov	eax,[b]
		mov	ecx,[r]
		mov	ebx,[g]
		push	eax
		push	ebx
		push	ecx
		mov	eax,[i]
		mov	ebx,[j]
		add	eax,320
		push	ebx
		push	eax
		call	DrawPoint	//  DrawPoint(i+320, j, r , g , b);
//		add	esp,0x14

		mov	eax,[b1]
		mov	ebx,[g1]
		mov	ecx,[r1]
		mov	[esp+16],eax
		mov	eax,239
		mov	[esp+12],ebx
		mov	[esp+8],ecx
		sub	eax,[j]
		mov	[esp+4],eax
		call	DrawPoint	// DrawPoint(i+320, 239-j, r1, g1, b1);
//		add	esp,0x14

		mov	eax,[b2]
		mov	ecx,[r2]
		mov	ebx,[g2]
		mov	[esp+16],eax
		mov	eax,[j]
		mov	[esp+12],ebx
		mov	ebx,639
		mov	[esp+8],ecx
		sub	ebx,[i]
		mov	[esp+4],eax
		mov	[esp+0],ebx
		call	DrawPoint	// DrawPoint(319-i+320, j, r2, g2, b2);
//		add	esp,0x14

		mov	eax,[b3]
		mov	ebx,[g3]
		mov	ecx,[r3]
		mov	[esp+16],eax
		mov	[esp+12],ebx
		mov	[esp+8],ecx
		mov	eax,239
		mov	ebx,639
		sub	eax,[j]
		sub	ebx,[i]
		mov	[esp+4],eax
		mov	[esp+0],ebx
		call	DrawPoint	// DrawPoint(319-i+320, 239-j, r3, g3, b3);
		add	esp,0x14

		dec	[i]		//     i--;
		jns	loop_x		//   } while ( i >=0 );
		dec	[j]		//   j--;
		jns	loop_y		// } while ( j >= 0);
	}
	return TRUE;
}

BOOL main(){
	unsigned int r,g,b;
	unsigned int r1,g1,b1;
	unsigned int r2,g2,b2;
	unsigned int r3,g3,b3;
	unsigned int R,G,B;
	unsigned int R1,G1,B1;
	unsigned int R2,G2,B2;
	unsigned int R3,G3,B3;
	int l[8];
	int i,j,k,x,y;

	R =0;	G =0;	B =0;
	R1=0;	G1=0;	B1=0;
	R2=0;	G2=0;	B2=0;
	R3=0;	G3=0;	B3=0;

	for(k=0; k<8; k++){
		l[k]=k*k/8;
	}

	for(j=0; j<120; j++){
		for(i=0; i<160; i++){
			for(k=0; k<8; k++){
				x=i-l[k]*-(160-i)/49;
				y=j-l[k]*-(120-j)/49;
				GetPoint(x, y, &r, &g, &b);
				R+=r;
				G+=g;
				B+=b;
				GetPoint(x, 239-y, &r1, &g1, &b1);
				R1+=r1;
				G1+=g1;
				B1+=b1;
				GetPoint(319-x, y, &r2, &g2, &b2);
				R2+=r2;
				G2+=g2;
				B2+=b2;
				GetPoint(319-x, 239-y, &r3, &g3, &b3);
				R3+=r3;
				G3+=g3;
				B3+=b3;
			}
			r =R /8; g =G /8; b =B /8;
			r1=R1/8; g1=G1/8; b1=B1/8;
			r2=R2/8; g2=G2/8; b2=B2/8;
			r3=R3/8; g3=G3/8; b3=B3/8;
			R =0;	G =0;	B =0;
			R1=0;	G1=0;	B1=0;
			R2=0;	G2=0;	B2=0;
			R3=0;	G3=0;	B3=0;
			DrawPoint(i,     j+240    , r , g , b );
			DrawPoint(i,     239-j+240, r1, g1, b1);
			DrawPoint(319-i, j+240    , r2, g2, b2);
			DrawPoint(319-i, 239-j+240, r3, g3, b3);
		}
	}
	return TRUE;
}
```

あなたの知らないWebサーバの世界

大和 哲/Yamato Satoshi

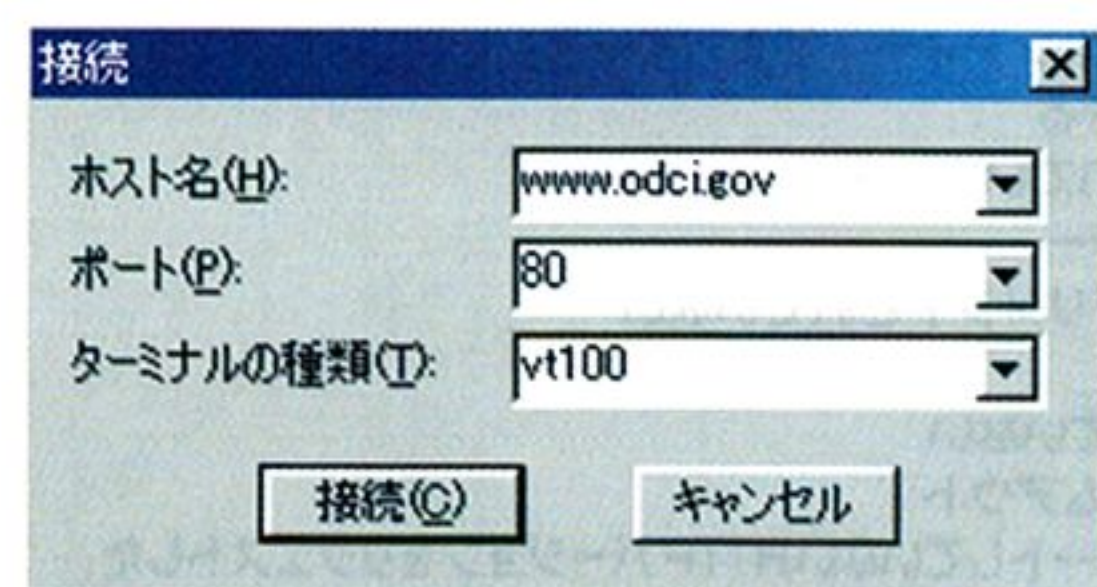
最近では個人でWebサーバを立てるのもさほど非現実な話ではなくなってきた。ここではWeb上でのデータのやり取りの基本とサーバプログラミングの初歩について解説する。Perlを用いたCGIでの基本処理を見てみよう。

■Webサーバのしくみ

ちょっと前に，Oh!XのWeb掲示板で「どこのプロバイダはどのWebサーバを使っているんだろう?」というのが話題になっていたことがあります。結局，そのときは「Windows用の，インターネット上のサーバソフトの名前がわかるソフトがあるよ」ということで話の決着がついたのですが，ホントはこれ，おかしな話なんですよね。だって，サーバソフトは常に私たちのPCにその正体を明かしているんです。Webサーバのしていることを見てみるとわかります。

ここでWebサーバのしくみを簡単に説明しましょう。複雑な話はざっくり飛ばします。インターネットでの通信にはTCPというプロトコルが使われていることくらい，わかっていればまぁ，いいでしょう（TCP/IPという名前が，Windowsのネットワークの設定でも出てくるはずです）。

さて，あなたがWebブラウザでサーバにアクセスすると，Webサーバはいつもさっと，反応してあなたにレスポンスを返してくれますね。そう，そのために，Webサーバソフトはじっと，サーバコンピュータの中で，あなたのブラウザからデータが流れてくるのを待ち続けています。

で，このTCPというプロトコルではデータを相手から受け取るための受け皿がいくつか並んでいます。これが「ポート」というもので，それぞれに番号がついています。なぜ1個ではなく，いくつもあるかというと，TCPプロトコルさえ確立できればインターネットを使ういろいろなことが1台のサーバ/クライアントでできるようになっているからです。

Windowsのtelnetを起動してみる

さて，このポート番号は，一応，どのソフトにこれ，と自由に割り振ってもかまわないのですが，一般的に割り振られている番号があります。なぜなら，サーバは誰からもデータを受ける可能性があるので，できるだけみんなにあわせたほうが，見にくる人がほかの場所と同じ設定でなんの苦労もなくデータの受け渡しができるからです。このポートのことをWell-knowポートといいます。日本語でいえば「よく知られているポート」。そのまんまですね。

このWell-knownポート，代表的なものではこんな数字が割り振られています。

FTP 21
TELNET 23
SMTP 25
HTTP 80
POP3 110

つまり，Webサーバは80番のポートにデータが入ってくるのをじっと待っているわけです。ちなみに，FTPのサーバソフト（デーモンというほうが一般的な気がしますが）は21番のポート，TELNETのデーモンは23番のポート，とほかのポートをそれぞれ，じっとデータがくるのを待っています。

さて，そして，ブラウザがサーバの80番ポートに「このページを読みたい」というリクエストデータを投げると，投げられたデータをWebサーバが受け取り解釈して，リクエストにあったホームページデータを投げ返します。このとき，クライアント側はどんな形式のデータが，どのような方法でほしいのか，受け取っても大丈夫なデータはどんな形式のものなのかなどのデータを，それに対して，サーバ側は実行結果はOKだったのか（NGだったらその理由は），データの形式は，サイズは，作られた日は，そしてデータの本体などを送りますが，それぞれ，自分の身元を明かすためなのか，自分がなんという名前のソフトであるかをお互いに明かします。たとえば，クライアント側は，

Mozilla/3.0Gold (WinNT;I)

つまり，自分がWindowsNT（Intel版）のNetscape Navigator 3.0Goldであるよ，と明かします。それに対してサーバ側も，

Apache/1.2.4
Netscape-Enterprise/2.01-p100
Microsoft-IIS/4.0

というように自分の正体を常に明かしているのです。

つまり，私たちがWebサーバがなんなのか知らないのは，常にブラウザに正体を名乗っているのに，ブラウザがそのメッセージを全然表示してくれないから，なんですね。もったいないですよねぇ。

■人間Webブラウザになる

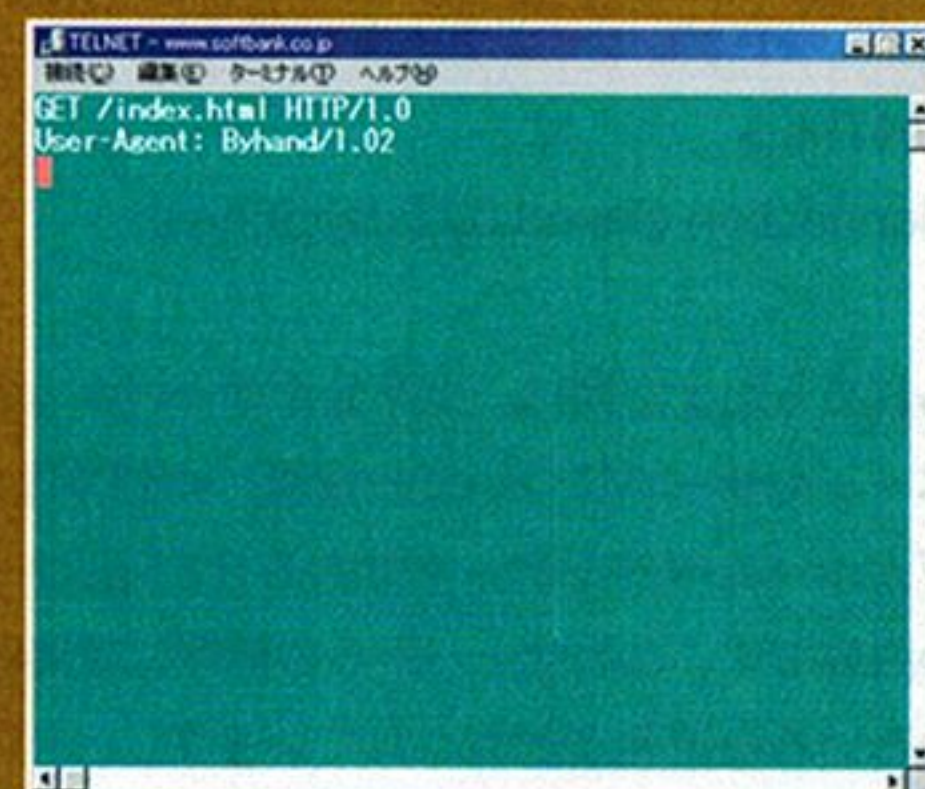
さて，先ほど「Webサーバのやっていることが見られれば，Webサーバが名乗っているのがわかる」と書きましたね。では，実際に見てみましょう。でも，Webブラウザでは表示されないメッセージをいったいなんで見たらいいというのでしょうか。実は，

Windows95/98マシンでのtelnetの注意

COLUMN

Windowsディレクトリにtelnetは転がっていますが，うまく使うにはコツがあります。まず，ポート番号は一覧の部分には直接数値を書き込んでください。ローカルエコーをOnにすれば，Windows98ではあとは問題ないでしょう。Windows95の場合は，さらに送信でCtrlコードを直接送る必要があります。すなわち，リターンキーを押す局面では，

Ctrl-M
Ctrl-J

を続けて押してください。それでちゃんと動作するはずです。

「サーバのメッセージを逐一見られる」
「自由にメッセージを入れられる」
「TCPポートを使っている」
とても都合のよいソフトがあります。TCP，Well-knownポートを使っているソフトのクライアントのひとつです。もったいぶってもしかたありませんね。そう，それはTELNETです。telnetクライアントソフトを起動したら，ホスト名とポートの80番を指定して接続してみてください。すると以下のような画面が出てきて，文字入力ができるようになります。

```
telnet> open localhost 80
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
```

キーボードから，ホームページデータがほしい，という意味の「GET」コマンド，GETするファイル名，それからHTTPの形式を入力します。

```
GET /index.html HTTP/1.0
```

それから，ブラウザのソフト名を名乗ります……まぁ，ソフトでなくて手作業なのでここは適当に，

```
User-Agent: ByHand/0.00
```

とでもしておけばいいでしょう。これでコマンド入力が終わったことを教えてあげればブラウザからのメッセージは終わりです。次の行はなにも書かないでリターンキーのみを押します。すると，Webサーバから直ちにサーバからのメッセージとWebページデータが送られてきて，telnetクライアントソフト上に次のようなメッセージが表示されるはずです。

```
HTTP/1.1 200 OK
  Date: Mon, 27 Jul 1998 23:36:24 GMT
  Server: Apache/1.2.4
  Connection: close
  Content-Type: text/html

  <HTML>
  <HEAD>
  <TITLE>Welcome Page</TITLE>
  <META content=text/html>
  <META content="HOTALL Ver.4.0W"
  name=GENERATOR>
  (以下，HTMLページデータが続く)
  </HTML>
  Connection closed by foreign host.
```

送られてきた文字列は，上半分がサーバからのこのページやサーバについての情報，<HTML>から下がホームページデータであるHTMLファイルの内容です。このサーバからの情報には興味深いいろいろな情報が記載されています。

```
HTTP/1.1 200 OK
```

この1行目の文字列は，送られてきたHTTPのバージョンと，サーバがデータを探した結果がどうだったかが示されています。「OK」だったわけですね。200という数字はOKに割り振られているリザルトコードです。サーバがリクエストされたファイルデータを探した結果はメッセージ1パターンにひとつのコードが割り振られています。404がFile Not Found（ファイルが見つからない）だといえば，ああ，あれ，という人が多いはずですね。

ほかには500番の「Internal Server Error」なんていうのもCGIを作る人にはお馴染みかもしれません。リザルトコードは200番台が成功，300，400，500番台が失敗になっています。主なリザルトコードを表に挙げておきます。

次の行の，

```
Date: Mon, 27 Jul 1998 23:36:24 GMT
```

はデータがサーバから送られた日時ですね。これは問題ないでしょう。

さて，次が問題のコードです。

```
Server: Apache/1.2.4
```

そう，ここでサーバが名乗りをあげてくれているわけですね。このサーバはApache Ver1.2.4。実はローカルでCGIのテスト用に作ったサーバマシンの（お古のマシンで簡単に作れます。うちのも使わなくなったDX2/50MHzマシンです）FreeBSD2.2.5に添付されていたフリーのWebサーバです。ちなみに，その下の，

```
Connection: close
Content-Type: text/html
```

表2　HTTP/1.0，HTTP/1.1のステータスコード（RFC1945,RFC2068より）

ステータスコードは３桁の数字からなっていて，HTTP/1.0とHTTP/1.1の間には上位互換があります。ステータスコードの最初の１桁はステータスコードの内容によって５つに分類されています。残りの２桁については分類分けは特にされていません（これはHTTP/1.0，HTTP/1.1とも共通です）。
ステータスコードの最初の１桁の分類は以下のようになっています。

- o 1xx: 情報 - リクエストを受け取った，継続プロセス
- o 2xx: 成功 - 動作の成功，受信済，了解など受け入れられた動作
- o 3xx: リダイレクション - - Further action must be taken in order to complete the request
- o 4xx: クライアントエラー - 実行できなかった（文法エラーなどを含む）
- o 5xx: サーバエラー - サーバがリクエストされたサービスに失敗した

ステータスコードの概要は以下のとおりです。

コード	名称	説明
"100"	; Continue	接続は継続しています（1.1のみ）
"101"	; Switching Protocols	上位プロトコルに移行しました（1.1のみ）
"200"	; OK	成功
"201"	; Created	作成成功（PUTメソッドなどで使われる）
"202"	; Accepted	受け入れられた
"203"	; Non-Authoritative Information	認証情報
"204"	; No Content	内容がない
"205"	; Reset Content	内容をリセット
"206"	; Partial Content	内容の部分のみ
"300"	; Multiple Choices	複数の選択肢があり送るべきデータを特定できない
"301"	; Moved Permanently	移動した
"302"	; Moved Temporarily	一時移動した
"303"	; See Other	よそを見よ
"304"	; Not Modified	修正されていない（If-Modifiedなどでリクエスト時）
"305"	; Use Proxy	Proxyを使え
"400"	; Bad Request	送られたリクエスト情報に問題
"401"	; Unauthorized	認証情報が必要
"402"	; Payment Required	（将来のためのリザーブ領域）
"403"	; Forbidden	アクセスが許可されていない
"404"	; Not Found	ファイルが見つからない
"405"	; Method Not Allowed	その方法でのアクセスは許されない
"406"	; Not Acceptable	受け取れないリクエスト
"407"	; Proxy Authentication Required	Proxyでの認証が必要
"408"	; Request Time-out	タイムアウト
"409"	; Conflict	コンフリクト（PUTなどの衝突）
"410"	; Gone	（404だがサーバが移動先を知っている）
"411"	; Length Required	ヘッダにLength:タグが必要
"412"	; Precondition Failed	Precondiction失敗
"413"	; Request Entity Too Large	リクエストエンティティが大きすぎる
"414"	; Request-URL Too Large	URLが長すぎる
"415"	; Unsupported Media Type	サポートされていないメディアタイプ
"500"	; Internal Server Error	サーバの内部エラー
"501"	; Not Implemented	このサーバにはインプリメントされていない
"502"	; Bad Gateway	ゲートウェイがおかしい
"503"	; Service Unavailable	サービスは行われていない
"504"	; Gateway Time-out	ゲートウェイのタイムアウト
"505"	; HTTP Version not supported	サーバがサポートしていないHTTPバージョンをリクエストした

※詳細についてはhttp://www.w3.org/からRFC1945，RFC2068を参照してください。

リスト1

```
#!/usr/local/bin/perl

#
# webview - view other web page on this web
#           by de.yamato 1998
#      e-mail de@sakuramail.com

use Socket;

require "cgi-lib.pl";
require "jcode.pl";
my @val;

&ReadParse(*input);

print &PrintHeader;
print'<META content="text/html; charset=charset=EUC-JP" http-equiv=Content-Type>';
print &HtmlTop("WebView test.");

print "<h2>WebView Result.</h2>\n";

@val = split(/&/,$input);

#16進数を元のキャラクタに戻す
foreach $i(0 .. $#val){
            ($name,$value) = split(/=/,$val[$i],2);
            $value =~ s/%(..)/pack(C,hex($1))/eig;
            $val{$name} = $value;
}

foreach $name(sort keys(%val)){
            printf "$name = %s <BR>\n",$val{$name};
}

print "<HR>\n";

# 指定されたホストの80番(web)ポートに接続する
TcpConnect(SOCK,$val{"host"},80) or &CgiError;

select(SOCK); $| = 1; select(STDOUT);

#GETコマンドを送信
&SendString(<<SENDSTRING);
GET $val{"file"} HTTP/1.0
Connection: close
Cache-Control: no-cache
Pragma: no-cache
Accept: */*
User-Agent: WebView/0.01

SENDSTRING

#受けた文字列を画面に
while(<SOCK>){
            &jcode'euc($_);
            print $_.'<BR>';
}

close(SOCK);

# HTMLの終末処理
print "</PRE>";
print &HtmlBot;

exit(0);
# end

# 応答コードの異常の場合
sub CgiError
{
            print "Cgi処理中にエラーが発生しました.<BR>";
            print &HtmlBot;
            exit(0);
}

# Socketに文字列を送信
sub SendString
            {
            local($_) = @_;

            # 行末はCR+LFに
            s/([^\r])\n/$1\r\n/g;
            print SOCK;
            }
# end SendString

#TCPの接続
#(「インターネットダイレクトアクセス」から…)
sub TcpConnect
            {
            my($socket,$host,$port,$sockopt) = @_;
            my($ipaddr,$result,$pack);

            $result = 1;
            $pack = caller;
            $socket = $pack . '::' . $socket;
            $port = getservbyname($port,'tcp') if( $port =~ /\D/ );
            &CgiError unless( $port );
            $ipaddr = inet_aton($host) or &CgiError;
            socket($socket,PF_INET,SOCK_STREAM,getprotobyname('tcp'))
                        or $Error = "ソケットがオープンできない",return 0;
            USESOCKET:
                        {
                        if( $sockopt ne '' )
                                    {
                                    setsockopt($socket,SOL_SOCKET,$sockopt,1)
                                                +or $Error = "ソケットオプションが設定できない
                                                ",$result = 0,last;
                                    }
                        connect($socket,sockaddr_in($port,$ipaddr))
                                    or $Error = "接続できない",$result = 0,last;
                        }
            # end USESOCKET:
            close($socket) unless( $result );
            $result;
            }
# end TcpConnect
```

はデータが送られたあとは接続が切れること，送られているデータはテキスト形式のHTMLファイルである，ということを示しています。接続が切れたことは，telnetクライアントに表示された最後のメッセージ，

Connection closed by foreign host.
(接続先のホストによって切断されました)

からもわかるでしょう。

サーバのメッセージの基本はこれですが，サーバやデータや接続方法によってほかにいろいろ情報が追加されることがあります。

このtelnetを使ってWebサーバと会話する方法，ひとつファイルをWebサーバからもらうたびにtelnetの起動からしなくてはいけないので，ちょっと面倒という欠点はありますが，Webサーバの動作を身をもって体感するにはいい方法ではないかと思います。機会があったら一度ご覧になってみることをおすすめします。リムネットのようにシェルをユーザーに開放しているプロバイダに入っているのであれば，そこからtelnetしてみる，という方法もありますしね。

■サーバソケットを使う

さて，先ほど「ポートはデータの受け皿である」と書きましたが，実際にはポートはデータを受けるだけではなく，サーバからデータを送ることもできます。つまりなろうと思えばサーバがほかのサーバのクライアントになることもできるわけです。実際にはセキュリティなどの観点から，一般ユーザーに開放しているプロバイダは多くありませんが，一部には開放しているプロバイダもあります。最近は専用線もOCNエコノミーなどで安くなってきたので，自分でWebサーバを作る方もなかにはいるでしょう。そういう方ならサーバのポートも使い放題です。もし，そういうチャンスに恵まれたら，一度はサーバプログラミングを体験してみてください。いままで見えなかったPCとインターネットの使い方が見えてくるはずです。

Webサーバ側でのプログラミングというと，一般のユーザーにとっては，やはりPerlを使ったCGIプログラミングが手近でしょう。ポートの使用が許されている環境であれば，CGIでサーバのTCPポートを使って，ほかのサーバにアクセスすることもできます。Perlで，TCPポートを使うには，

use Socket;

でSocketライブラリを使います。これでTCPポートをオープンして，そこから，先ほどのGETコマンドをサーバに送り，サーバから送られてきたメッセージを表示すれば，いちばん単純なサーバを使ったWebクライアントのできあがりです。

CGIでは標準出力で文字を出力すると，それがそのままWebブラウザで表示されますから，よそのサーバから受けてきたデータをそのまま標準出力に流すCGIを作れば，CGIによるWeb中継ページのできあがりなわけですね。せっかくですから，サーバからのリザルトコードも正直に出力するWeb中継CGIを作ってしまいましょう。

さて，実は筆者もきちんと完全に理解しているわけではないのですが(実はサンプルプログラムのTCPソケット接続部分は，参考文献のサンプルほとんどをそのまま使用しています)，TCPポートをクライアントとしてアクセスするにはだいたい以下のような手順で行います。

1) Socketをオープン
2) connectでサーバのポートに接続
3) データの送受信
4) Socketのクローズ

Perl5の場合，一度Socketをオープンしてサーバへの接続が確立されてしまえば，ハンドルを使って，

While (<SOCK>)

で1行ずつデータを読み込み，

print SOCK

でデータの1行送信ができます。

ということで作ってみたのが，リスト1のwebview.CGI。ほかのWebサーバの内容を表示するCGIプログラムです。上の手順でWebサーバにアクセスし，

GET（ファイル名）HTTP/1.0
Cache-Control: no-cache
Pragma: no-cache
Connection: Close
Accept: */*
User-Agent: WebView/0.01

と，GETコマンドを発行しています。GETメソッドコマンドに続く文字列はクライアントからサーバに渡す情報です。GETコマンドの後ろには本来，このようにクライアントから，接続方法とどんな情報がほしいか，などの情報（リクエストヘッダ）とどんなデータなら受け取れるかを示すエンティティヘッダを送ることになっています。リクエストヘッダの種類については，「表：リクエスト，リザルト情報で使われるコード」に主なリクエストヘッダの要素を挙げておきました。

これらの書式も含めてHTTPのプロトコルなどについては，HTTPやHTMLの標準化をすすめるW3Cというコンソシアムから定義を決めた文書が出ています。一世代前のHTTPプロトコルであるHTTP/1.0の場合はRFC1945，最新のプロトコルであるHTTP/1.1はRFC2068という文書になっています。もし，興味があって，かつ英語の文書を読む気力のある方はインターネットから見ることができますので，そちらを参照してみるといいでしょう。

RFC1945は，

http://www.w3.org/Protocols/rfc1945/rfc1945

RFC2068は，

http://www.w3.org/Protocols/rfc2068/rfc2068

にあります。ちなみに，初めてご覧になるのであればまずRFC1945を読んでみてからRFC2068をお読みになることをおすすめします。Connectionをcloseで指定さえしておけば，HTTP/1.0のリクエストで大半の部分こと足りますし，全体にRFC1945のほうがかなり簡潔で読みやすい，というのがその理由です。というか，RFC2068は全体でもかなりの量ですが，1つひとつの説明がくどくて，いきなり読むとイヤになると思います。私もそうでしたから。

さて，CGIに話を戻しますが，webview.htmlファイルをブラウザで表示させて，サーバ名のアドレスとファイル名を入力し，「送信」ボタンをクリックすると，そのページのヘッダとページの内容が表示されます。サーバ名は「http://」などは入れず，「www.softbank.co.jp」とマシンのアドレスのみを入れてください。またファイル名は必ず「/」が先頭に入るパス名を入力します。つまり，「http://www.softbank.co.jp」の内容を見たかったら，アドレスに，

www.softbank.co.jp

ファイル名に，

/

を入力します。

http://www.softbank.co.jp/softbank/publishing/dosvstart/netx.htm

であれば，アドレスに，

www.softbank.co.jp

ファイル名に，

/softbank/publishing/dosvstart/netx.htm

を入れます。

正しくサーバに接続されると，

file = /
host = microsoft.com

とファイル名，ホスト名が表示されて，その下に，

HTTP/1.1 200 OK
Server: Microsoft-IIS/4.0
～（以下略）

と，サーバのリザルト情報，ホームページの本体が表示されるはずです。ページが見つからない，などのエラーのときもサーバのレスポンスがそのまま表示されます。

作者のWebページ，

http://deyamato.fsn.net/webview.html

にもサンプルが稼働していますので，よろしければ試してみてください。なお，このCGIには日本語処理支援ルーチンのjcode.pl，CGIプログラムの支援ルーチンのCGI-lib.plを使用していますので，お使いになる前にそちらを先に入手しておいてください。どちらも有名なルーチンですから簡単に手に入るでしょう。HTMLファイル，CGIとも漢字コードは強制的にEUCに変換して出力していますので，文字化けした場合には表示をEUC漢字に変えてみてください。また，このプログラムは，perl5へのパスが/usr/local/bin/perl，HTMLのディレクトリから見てCGIプログラムが../CGI-binディレクトリにあることになっていますのでお使いの環境にあわせて該当する部分を書き換えてください。当然，サーバがUNIX上でperl5が動き，socket.pmにアクセスできる設定になっていないとこのCGIを使うことはできません。

ちなみに，このWebサーバへのアクセスに，会社などでProxyサーバを使ってアクセスしている場合があると思います。残念ながら，このサンプルプログラムをそのまま使うことはできませんが，実はProxyを経由したアクセスもソースのわずかの書き換えでできます。

Proxyを使ったWebサーバへのアクセスは，proxyなしのWebサーバとほぼ同じ手順で行えます。proxyなしのWebアクセスでは，Webサーバに直接アクセスしましたが，Proxyを間に通す場合，まず，Proxyサーバの指定されたポート（8080が多いと思いますが）をオープンします。それから，ホームページデータを受けるには「GET」コマンドを発行しますが，通常のWebサーバアクセスではファイル名を指定していたのを，ここでアクセス先のURLをフルパスで伝えます。telnetで確認できる環境があれば確認していただきたいのですが，つまり，http://www.softbank.co.jp/softbank/publishing/dosvstart/netx.htmにアクセスするのであれば，

GET http://www.softbank.co.jp/softbank/publishing/dosvstart/netx.htm HTTP/1.0

というコマンドを送ればいいわけです。

で，先ほどのCGIの場合は，

TcpConnect(SOCK,$val{"host"},80)
or&CGIError;

という行でTCPソケットの80番ポートを指定して接続をしていますので，

TcpConnect(SOCK,$val{"host"},8080)
or&CGIError;

表2　リクエスト，リザルト情報で使われるコード

メソッド	
GET	データを受け取る
HEAD	GETで受け取るデータのコンテンツ以外の部分を受け取る
POST	データを送る
PUT	データを送る
DELETE	データの削除
TRACE	データ経路を知る
ヘッダ	
Cookie:	Netscapeクッキー
If-Modified-Since:	これ以降の更新日だったらリクエストを実行せよ
If-Unmodified-Since:	この日以降に更新されなかったらリクエスト実行
From:	（CGIの場合は作者の）メールアドレス
Last-Modified:	最終更新日時
Location:	対象のURL
Pragma:	Proxyへのリクエスト
Referer:	参照元URL
Server:	サーバ名
User-Agent:	クライアントソフト名
エンティティヘッダ	
Accept-Charset:	受け取り可能なキャラクタセット
Accept-Encoding:	受け取り可能なエンコード方法
Accept-Language:	受け取り可能な言語
MIME-Version:	MIMEのバージョン

とお使いのproxyの指定ポートをアクセスするようにしてください。それから，アドレスにproxyサーバのアドレス，ファイル名の部分に見たいWebページのフルアドレス（http://www.geocities.co.jp/SiliconValley/6860/OhX/geobook.htmlとか）を入力することで，proxyなしでWebアクセスをしたときと同様にリザルト情報付きでHTMLファイルが表示されるはずです。

■いろいろ応用が考えられるよね

さて，WebサーバからSocketを使ってTCPポートでいろいろなことができる……，となるといろいろな応用が考えられると思います。たとえば，新作情報付きリンク集。先ほどのtelnetやwebview.CGIを試された方はわかると思いますが，Webサーバの中には，ページのリクエストをすると，サーバのレスポンスコードの中に，

Last-modified: Wed, 15 Jul 1998 09:07:22 GMT

とページの最終更新日を教えてくれるものがあります。Webブラウザのスタートページを，自分の好きなものを集めた自作のリンクページにしている人，というのは結構多いようですが，このCGIを組み合わせれば，リンク先のページの最終更新日もわかる，というわけです。

で，WebサーバからWebページデータを取ってくるためには「GET」コマンドを使いましたが，このほかにもサーバコマンドがいくつかあります。主なものでいうと，データを送るための「PUT」コマンドとか。先ほどの「GET」コマンドで送られてくるリザルト情報だけをもらって，データ本体を送らないようにさせる「HEAD」コマンドなんていうのもあります。ホームページ丸ごと取ってくるGETではなくて，「HEAD」というコマンドを発行すれば，GETでクライアントが得られるヘッダだけを取ってくることもできます。

ということで作ってみたのがHEADコマンドを利用して，リストファイルに登録されているホームページのリンクと同時に，それぞれのページの更新時刻を表示するCGIです（時刻はGMT，つまりグリニッジ標準時で表示されていますのでJSTから9時間遅れになります。CD-ROMに収録）。

テストとして，リストにOh!X関係のページのリストを作り（addr.datという名前のテキストファイルです），作者のページ，

http://deyamato.fsn.net/CGI-bin/webday.CGI

から実行できるようにしてみましたが，これだけでもどれが更新されたかわかってなかなか便利です（こちらは，addr.datというファイルにアクセスするページのURLとその説明を書くようになっていますが，http://から始まるフルパスで書いてください。ちゃんとアドレスを解析して使うようになっています）。さらに新しく更新された順にソート表示でもできるようにすればさらにおいしいでしょう。ぜひ改造してみてください。ちなみに，実は，Date:やLast-modified:での日付表示方式は本来

Wed, 15 Jul 1998 09:07:22 GMT

と表示されるのですが，古いWebサーバソフトを使っている場合などは，

Wednesday, 15-Jul-98 09:07:22 GMT

と違っていることもあるので，ソートができるようにするには，その辺りも改良が必要になるはずです。もっとも，そもそも，Last-modified:にしても，Size:などにしても必ず出力されるという保証はどこにもない（実際，最初のtelnetの例ではどちらもない，非常にシンプルなレスポンスでしたね）ので，本当に本格的にどのページでも必ず更新をチェックできるようにしようと思ったら，どちらも取れなかった場合の更新を知る手段を考えなくてはならない，という根本的な問題もあったりするのですが。

ちなみに，ほかの応用例として，筆者は，「Webページ上でメールボックスのメールを見るCGI」というのも作ってみました。先ほどのサーバのWell-knownポートの中に，

POP3 110

というのがありましたね。これが，メールを受信するためのポートです。Webページ表示CGIと同じようにCGIでWebサーバのSocketをオープンして，メールサーバのPOP3サーバに接続し，メールを読んでくる。シンプルなテキストのメールのみで考えるのであれば，作業の流れ自体はほとんどWeb表示CGIと変わりません。ただ，POP3はパスワードを入れなくてはならないので，さすがにホームページ中でパスワードを入れなくてはいけないようなプログラムはまずかろう，ということで公開はしていませんが。SMTPはさすがに使うことはないでしょうが（割とプロバイダでもsendmailが使えるところが多い），Well-knownポートにあるものでいうと，FTPポートを使うこともできるでしょう。

いまはCGIというと，チャットと掲示板が圧倒的（この2つも私は決して嫌いではないですが）でしたが，WebサーバのTCPポートが使えるとなると，それだけでない使い方もいろいろ楽しめるのではないかと思います。おそらくいろいろここまで使わせてくれるプロバイダ/ホスティングはあまりないでしょうが，それでも探せばないわけではありません。なに，昨今は，24時間接続だってまったくアマチュアの手の届かない金額ではないですから（OCNエコノミーは偉大ですね），いっそのこと，常時接続なインターネットWebサーバを自分用に作ってしまう，というのも気合次第では不可能ではありません。会社のイントラネットで，ちょっと古くなったPCにPC用のUNIXでも入れてみる，というのもテでしょう。Linuxはあまり知らないのですが，FreeBSDなどはフリーでUNIXもWebサーバも，今回私が書いたようなプログラムをSocketを意識しないで書けるライブラリのパッケージなど，この手の用途には足りないものがないくらいの充実しています。というか，FreeBSDではOSのインストール時にキーひとつでOSと同時にインストールできるパッケージというもので，すでにSocketを意識しないでもPerlからほかのサーバをアクセスできるライブラリまで一式揃っています。CGIのテスト用としてもこういうOSで自分専用のサーバマシンを作ってしまうのはおすすめです。

■気配りのススメ

さて，ここまで書いておいてなんですが，Webサーバというのは，ユーザーみんなが使うもの。うまく使えばいろいろ便利な使い方もできるわけですが，逆になにか問題を起こせば，ユーザーみんなに迷惑がかかります。皆さんのプログラムが大災害の元凶にならないように細心の注意を払って使うようにしてください。先ほどもちょっと書きましたが，この手のプログラムのテストはまず，ローカル環境で行うことをすすめます。注意しなくてはいけないものの代表的なものは，負荷，そしてセキュリティホールです。作ったプログラムが負荷がかかりそうかどうかは手元にサーバがあればすぐにわかります。

それから，先ほどの更新日チェックCGIなどは，本当は「robot.txt」というファイルを見てちゃんと判断すべきです。Webサーバでは，なにかの事情があって，検索エンジンなどにひっかかりたくない，などというときに「robot.txt」というファイルを作ってアクセスをコントロールすることができるようになっています。半自動の更新日チェックがこのようなアクセス制御に従うべきかどうかはちょっと微妙な気もしますが，サーバにとっては歓迎されない客である可能性も常に考えていなくてはなりません。

公共のWebサーバはみんなで使うものです。「使わせてもらっている」ぐらいの感覚でいたほうがいいかもしれません。そして，現状では，どちらかというとちょっとイレギュラーな部類に入る使い方をこれでしているわけです。周りには十分気をつけて実行しましょう。

参考文献：
Perl徹底活用インターネットダイレクトアクセス（情報管理社）・中島靖
Webクライアントプログラミングwith Perl（オライリー・ジャパン）・Clinton Won著 法林浩之監訳 須田隆久訳

Direct3Dを使うまで

菊地 功/Kikuchi Isawo

とにかく，いまや3D全盛の時代だ。ある意味でWindows上でのリアルタイム3Dはもっとも盛んに行われている分野だといえるだろう。その中核となっているのがDirect3Dだ。特にRetained Modeでは，基本的な概念さえ理解すれば，驚くほど簡単に3Dオブジェクトを表示することができる。そして，3Dアクセラレータがあれば，かなり複雑なものでもリアルタイム処理が可能になるのだ。ここではDirect3DのRetainedModeを中心に，考え方の基本を探っていってみよう。

■D3Dへ至る病

数年前まではワイヤフレームだポリゴンエンジンだと喜んでいたというのに(それはそれで素晴らしいものだったが)，技術の革新とは恐ろしいもので(この業界では死語となるくだりだが)，テクスチャ当たり前，グローシェーディング，バイリニアフィルタ，フォグなどを平気で搭載し，秒60フレームの3Dポリゴンシステムが当たり前になっている。

こういうものを最初に世に知らしめたのは，なんといってもセガ・エンタープライゼスの「バーチャファイター」だろう。あの衝撃はいまでも忘れない。当時はまだフラットシェーディングであったが，それでもその画面の美しさ，動きのスムースさに心奪われたものだ。それまではゲームといえばスプライトを用いた2Dが主流で，その頃格闘ゲームの不動の地位を築いていたカプコンの「ストリートファイターII」シリーズと比べると，まさしく別世界であった(こちらはこちらで徹夜で昇竜拳や波動拳の特訓をしたものだが)。

しかもそれが，多少パワー不足(というよりプログラムの未熟さ)ではしょられた部分もあるが，家庭用ゲーム機，セガサターンに移植されてしまったときは，さらなる驚きであった。数百万円からするアーケード基板ならいざ知らず，2〜3万円程度の「おもちゃ」でこんなことができる時代がきたのだ，と。

ではPCはどうだったかというと，あい変わらず「パソコンは仕事のためのもの」という思想が根底に流れていたため，WinGというゲーム用のライブラリはあったものの，そういったポリゴンエンジンなどとはほど遠い存在であった。コンシューマレベルで，初めて3Dエンジンが搭載されたのは，Matroxのビデオカード，Millenniumだろう(編注：その前のImpression Plusです)。

このカードは，CADなどのリアルタイムレンダリングを目的としたビジネス向けの製品だが，画面の美しさと2Dのパフォーマンスで人気を博したのは記憶に新しい。残念ながら当時はまだ3Dの規格などはなく，このMillenniumの3Dエンジンは専用に書かれたアプリケーションでしか発揮することはできなかった。また，テクスチャに対応しておらず，いまとなってはその3D性能は申しわけ程度である。筆者もこのMillenniumをいまもなお愛用しているが，3Dエンジンはほとんど使われることはなかった(とはいっても，2Dはいまなお高性能な部類に入る)。

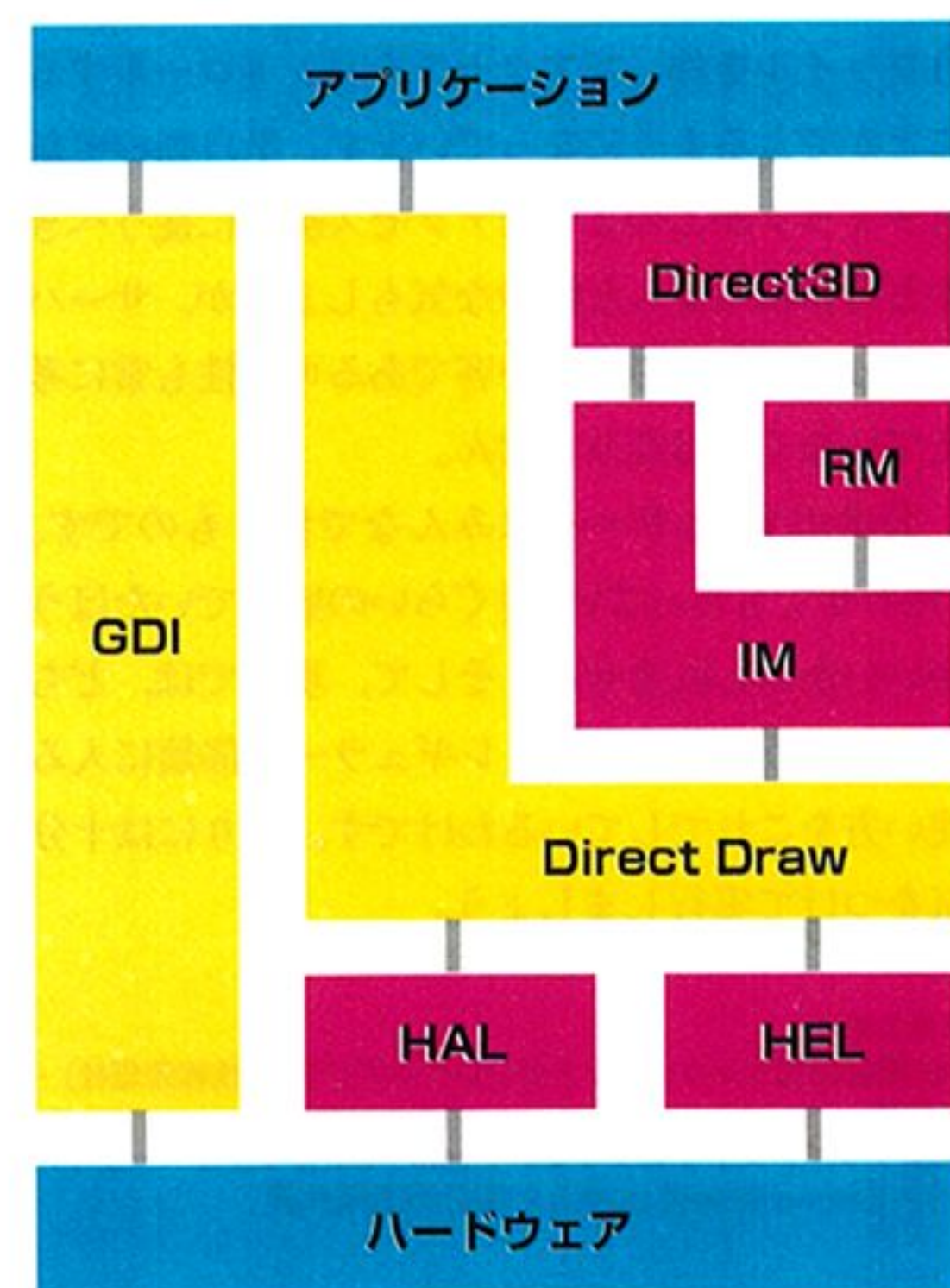
ゲームをターゲットとした3Dカードとしては，nVIDIAのnV1(およびその互換チップ)を搭載した，ダイアモンド・マルチメディア・システムズのEDGE 3Dがはしりである。サウンド機能も1枚のカードに搭載したマルチメディアカードで，奇しくもバーチャファイターがバンドルされていた。が，このカードはゲテモノ扱いされることが多く，あまり受け入れられなかったようだ。そのあとである。MicrosoftがWinGをさらに発展させたようなDirectXを発表し，3Dにも対応して一躍脚光を浴びた。それをきっかけに，各社から3D対応カードが次々と発売されて，今日に至っている。

では，なぜDirectXなのか。実際のところ，DOOMに代表されるように，DOSではポリゴンゲームの歴史は決して浅くはない。確かにそういったポリゴンエンジンはソフトウェアメーカーが各々持っていて，一般のユーザーが利用できないというのもあるだろう。だがそれだけではない。たとえばDOSの場合，多少VGAのファンクションを利用するにしても，基本的にジオメトリ演算をしたり，ビデオメモリに書き込んだりするのはCPUだ。ビデオカードが3Dに対応していても，だ。なぜなら3Dのファンクション(だけでなく，2Dのファンクションも)はカードによって異なるからである。これは規格がなかったというのもあるが，やはりチップによって内部構造や機能が異なるために，すべて統一というわけにはいかなかった。

したがって，DOSからこういった3Dエンジンを利用しようと思ったら，カードごとに別のコードを用意してやらなければならない。これは膨大な量に及ぶであろうことは，想像に難くない。もしそれが可能だとしても，そのあとに発売されたカードについては，動作しないことになる。これを回避するためのDirectXなのである。DirectXではいくつかのファンクションを規格化し，それらを利用するAPIをユーザーに提供する。また，チップメーカーやビデオカードメーカーは，その規格に準じたデバイスドライバを製品に添付したり，配布することで，そのカード上でDirectXが動作することを保証する。これによって，DirectXを用いて作成されたアプリケーションは，最新のビデオカードでも動作することが保証されるのだ。

ただ，DirectXを用いなくても，Windowsを(ネイティブな画面モードで)動作させるためにはビデオドライバが必要なのはご存じのとおりだ。ではなにが違うのだろう。Windowsのビデオ周りのAPIは，GDIという「抽象的な」オブジェクトを経由する。抽象的なAPIにすることによって，ユーザーからの描画命令を簡略化することができるからだ。そのためのインタフェイスがデバイスコンテキストである。

しかし，このような抽象的な命令では，ビデオドライバが命令を解析し，実際に実行に移すまで，少なからぬタイムロスが生じる。Wordで文章を書いているだけならいざ知らず，ミリセカンド単位での描画を必要とするゲームでは，このロスはばかにならない。そこでDirectXでは，より具体的で低水準なAPIを規定することで，このロスを少なくしている。これにより，GDIよりも高速で，よりゲームに即した描画が可能となる。

図1　描画へ至る経路

■DirectXによる開発

ところで，ここまでビデオの話しかしてきていないが，DirectXにはDirectSoundやDirectInputなどといった，ゲームに必要なインタフェイスがひととおり揃っている。もちろんそれらを利用すれば，Windows標準のAPIよりも高速に処理を行える。が，ここでは3Dのインタフェイスであるでirect3Dを中心に解説を行っていく。

Direct3DはDirectX2から提供されており，原稿執筆時点の最新版DirectX5ではいくつかのインタフェイスが拡張されている。本誌が出ている頃にはDirectX6が発表されているはずだが，ここではX5を基本とする。

DirectXでプログラミングするためにはDirectX SDKが必要だが，これは付録CD-ROMに収録してある。また，開発環境としてはVisual C++5.0を前提に話を進めていくので，まずはそれらをインストールしてもらいたい。インストールが済んだら，DirectX SDKのインクルードとライブラリにパスを通さなくてはいけない。VC++5.0には最初からDirectX3 SDKが含まれており，これをこのままにしておくと，DirectX3 SDKのインクルードとライブラリを先に見に行ってしまうので，削除しなければならないのだが，そのファイルを探し出すのは面倒なので，X5のファイルで上書きしてしまうのが手っ取り早い。具体的には，VC++をC:¥DevStudio¥VC，DirectX5 SDKをC:¥DXSDKにインストールしたのであれば，

C:¥DXSDK¥SDK¥INC を C:¥DevStudio¥VC¥INCLUDE
C:¥DXSDK¥SDK¥LIB を C:¥DevStudio¥VC¥LIB

にコピーしてしまえばよい。

■DirectDrawとDirect3D

ところで，Direct3Dは単独で存在するのではなくて，実際には2DのインタフェイスであるDirectDraw上に載っかっている。したがって，まずはDirectDrawから簡単に説明しよう。DirectDrawは2Dグラフィックの命令セットで，その内容は特に目新しいものではない(当然GDIに比べれば高速だが)。カラーキー(クロマキーといったほうが馴染みがあるかもしれない)による色抜きのスプライト(エミュレーション)，ビデオメモリへの直接アクセスなどといったことが可能だが，アニメーション時にもっとも重要となるのはフリップ機能だ。たとえば，複数の画面を次々表示して，アニメーションさせる場合を考えてみよう。

このとき，単に画面にピクセルを流し込んでいったのでは，上半分が次の絵，下半分が前の絵という状態が瞬間的に見えてしまい，非常に見苦しい。2枚の絵が酷似していればそれほど気にならないが，動きの激しいときや場面の切り替え時にはちらついてしまう。また，仮に画面の書き換えが非常に高速で，書き換え時間を無視できたとしても，好きなときに勝手に書き換えたのでは同じ現象が生じる。これは「走査線」というものに関係する。

ディスプレイを見ると，画面全体が発光してウィンドウなどを表示しているように見えるが，これは人間の目の残像現象でそう見えているだけで，実は画面上の1点を1列ずつ順番に発光させていっているだけなのだ。これが走査線だ。ブラウン管の原理を知っている人はおわかりだろう。この走査線は画面の左上から右方向へ進み，右端へくるとひとつ下の左端に戻る。このように走査していき，右下までくると，また左上に戻る。この右下から左上に戻るまでの時間を垂直帰線期間という。

ビデオドライバの設定などで，水平周波数60kHzなどというのは，1本の走査線が画面の左端から右端に到達するまでの時間を示しているのだ。さてここで，走査線が画面の半分くらいまできたときに画面を瞬間的に書き換えたとしよう。すると，画面の上半分は以前の状態が残像的に残っており，下半分は新しい画面が表示されることになる。これにより，上と下が反対だが，やっぱりちらつきが出てしまうわけだ(テアリングという)。こういった現象を避けるには，垂直帰線期間内に画面書き換えを済ませなければならないのだ。

Xユーザーだった方にはいまさらなにをいってるんだと思われる方も多いだろうが，Windowsではそういった基本的なことをする手段がそもそもなかったわけだ(垂直帰線が見えない)。

さて話は戻ってDirectDrawだが，DirectDrawでは「サーフェス」という，いわゆる画面バッファを複数取って，フリップによってそれらをハードウェア的に切り替えて表示することができる。

サーフェスをビデオメモリに確保しておけば，内容を転送するのではなく，画面に表示するメモリ領域そのものを切り替えてしまうので，切り替え時間は限りなくゼロに近い。しかもフリップは必ず垂直帰線期間に行われるので，ちらつきはまったくなくなる。プライマリサーフェスとバックサーフェスの2つを取って，バックサーフェスに必要なものを書き込んでからフリップする，といった手法が一般的だ。

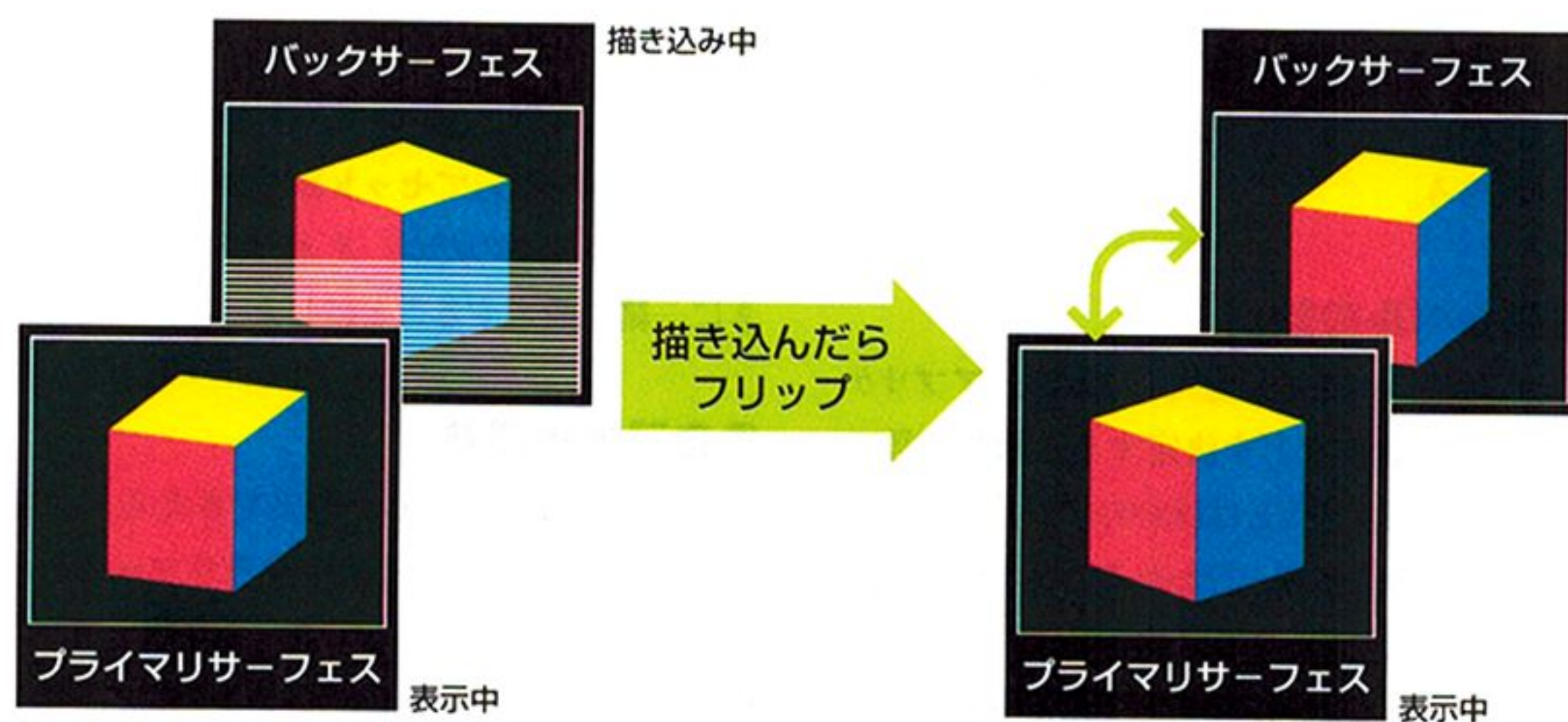
注意が必要なのは，フリップするとプライマリサーフェスとバックサーフェスが入れ替わること，フリップはフルスクリーンモードでしか使えないことだ。こういった手法を使ったサンプルDDSampleがCD-ROMに収録されているので，DirectDrawの使い方などとともに見ていこう。ワークスペースDDSample.dswをDeveloper Studioで開いてもらいたい。

■DirectDrawサンプル

さて，このサンプルはMFCを使っていない。したがって，VC++のほうの記事でもあったように，ウィンドウクラスの登録やメッセージポンプなどは自前でやっている。とはいえ，DirectDrawをフルスクリーンモードで使うのに特に特殊なメッセージを取ることもないので，基本的な部分さえ覚えてしまえば，あとは自分でプログラムを組むときに，そこからコピーペーストすればよい。

●メッセージポンプ

DDSample.cppの中の，WndProc()とWinMain()がそれだ。DOSのCのプログラムが必ずmain()関数から入るように，WindowsのアプリケーションはまずWinMain()関数が呼ばれる。

図2　フリップするとプライマリとバックサーフェスが入れ替わる

ここではウィンドウクラスを登録し，ウィンドウを表示したら，ループを回してメッセージを見に行っている。この中で，

```
// DirectDrawの初期化
if( !InitDDraw( hWnd, hInstance,
nCmdShow ) ) return FALSE;
// サーフェスの初期化
if( !CreateOnScreenSurface( hWnd ) )
return FALSE;
```

と，メッセージポンプ内の，

```
if( bActive ) Render();
```

を除いた部分は，Windowsのプログラムの決まり文句である。詳しくは説明しないが，興味があればヘルプなり関連書籍を読むなりしてほしい。

WndProc()のほうは，ウィンドウがなんらかのメッセージを受け取ったときに呼ばれる関数である。こちらもこの形が基本で，ハンドルしたいメッセージをcase文で増やしていくことになるが，ここでは必要最小限のものだけを拾うことにする。では流れに沿って説明しよう。

●初期化

まずはInitDDraw()関数だ。ここで行っているのは，DirectDrawオブジェクトの作成，協調レベルの設定，画面モードの設定の3点だ。DirectDrawオブジェクトは，すべてのDirectDraw関連のオブジェクトの大本である。DirectDrawはCOMモデルで……なんて頭の痛い話は置いといて（実際COMがなんなのかわからなくてもさしたる支障はないし，取りたてて騒ぐほどのものでもない），とにかくDirectDrawCreate()というAPIによって，DirectDrawのインタフェイスが取得できることを覚えておけばよい。

この関数はddraw.hで宣言され，ddraw.libで定義されているので，自分でプログラムを組む際にはインクルードとリンクを忘れないように。戻り値のDD_OKというのは，DirectDraw関係の関数で共通に利用される定数だが，要するに正常終了というわけだ。

もしDD_OKが返ってこなかったら，処理は正常に終了しなかったということだが，DirectDrawCreate()関数が成功しない場合というのは，DirectXがインストールされていない状態と考えていいだろう。

●協調レベルの設定

続いて協調レベルの設定だ。これは，アプリケーションの優先度などを決定する。第2引数のDDSCL_EXCLUSIVEはほかのアプリケーションよりも優先的に処理が行われること，DDSCL_FULLSCREENはフルスクリーンで実行されることを示し，この2つはセットで指定されると思ってよい。

反対にウィンドウモードで動作させるときにはDDSCL_NORMALを指定する。要は，フルスクリーンモードでは，ほかのウィンドウやアプリケーションの存在を忘れて，画面表示もCPUパワーも勝手に好き放題使わせてもらいますよ，ということだ。

ところで，IDirectDrawはインタフェイスだが，lpDD->SetCooperativeLevel()をC++ではなくCでやろうとすると，

```
lpDD->lpVtbl->SetCooperativeLevel( lpDD, hWnd,
DDSCL_EXCLUSIVE|DDSCL_FULLSCREEN );
```

として，lpVtblを経由しなければならない。でもって，第1引数のlpDDというのがthisポインタだ。なんとなくカラクリが見えてきたような気がするが，Cでこのような冗長な記述をするよりは，C++のほうがシンプルでよいだろう。

●画面モードの設定

話が横道にそれたが，最後は画面モードの設定である。lpDD->SetDisplayMode()で画面の幅，高さ，カラービットを指定している。640×480の8 Bits/Pixel，つまり256色モードということだ。ここまでで画面の初期化は終わり。この時点で画面は真っ暗になり，フルスクリーンの指定したモードになる。

次はサーフェスの初期化だ。先ほども説明したように，まずはプライマリサーフェスとバックサーフェスを構築する。この辺りもパラメータを含めてお決まりの手である。バックバッファを1枚指定してプライマリサーフェスを作り，バックサーフェスをアタッチしている。

バックサーフェスはプライマリサーフェスの作成時に同時に作成されるので，特にCreateSurface()で作成する必要がないというのがミソだ。あとは，画面の描画に必要な材料を作成する。今回は背景の中を泳ぐエンゼルフィッシュをスプライトで表現させているので，背景とエンゼルフィッシュのイメージが必要だ。

これらもまたサーフェスを構築してそこに保存しておくのだが，こういった直接表示されないバッファ的なサーフェスをオフスクリーンサーフェスという。

あとは，今回は256色モードなので，とりあえずパレットを作ってセットしておく。パレットの中身はまだ空なのだが，イメージをロードしたときに一緒に設定することにしよう。

●自己修復関数

ここまで必要なものが構築できたら，今度はRestoreSurface()でその中身をリソースからロードする。関数まで作ってきっちりサーフェスの構築とリソースのロードを分けたのには意味がある。現在はフルスクリーンモードで動作するアプリケーションを作っているが，動作中にAlt+Tabキーなどを押して，Windows画面に戻ったらどうなるだろうか。画面を占有するということで了解を受けて動作していたこのアプリケーションの画面は破壊され，サーフェスのインタフェイスもずたずたになってしまうのだ。

こうなった場合のために，サーフェスには自己修復するRestore()関数が用意されている。たとえばプライマリサーフェスの場合，lpDDSPrimary->Restore()といった具合だ。ただこれで修復されるのはメモリの再確保までで，内容までは保証されない。そこで，そのあとで内容をリロードさせる必要があるのだ。つまりRestoreSurface()はそれと兼用の関数なのだ（修復はWndProc()のWM_ACTIVATEAPPメッセージから呼んでいる）。ここで重要なのは，サーフェスへの書き込み方法だ。

サーフェスはGetDC()によってデバイスコンテキストを取得することができるので，そのデバイスコンテキストを使ってビットマップを選択したメモリデバイスコンテキストからBitBltでコピーする。リソース内のビットマップは，背景と魚のアニメーションを1枚にしてあるので，それらの座標を考えてサーフェスにコピーしてやっている。

サーフェスからデバイスコンテキストを取得する方法は，こういった用途のほかに画面にちょっと文字を表示したいといった場合にも便利なので，覚えておくとよいだろう。最後はカラーキー，すなわちスプライトで透過させる色を設定している。とりあえずこれで初期化部分は終わりだ。実際の描画はRender()関数内で行われる。

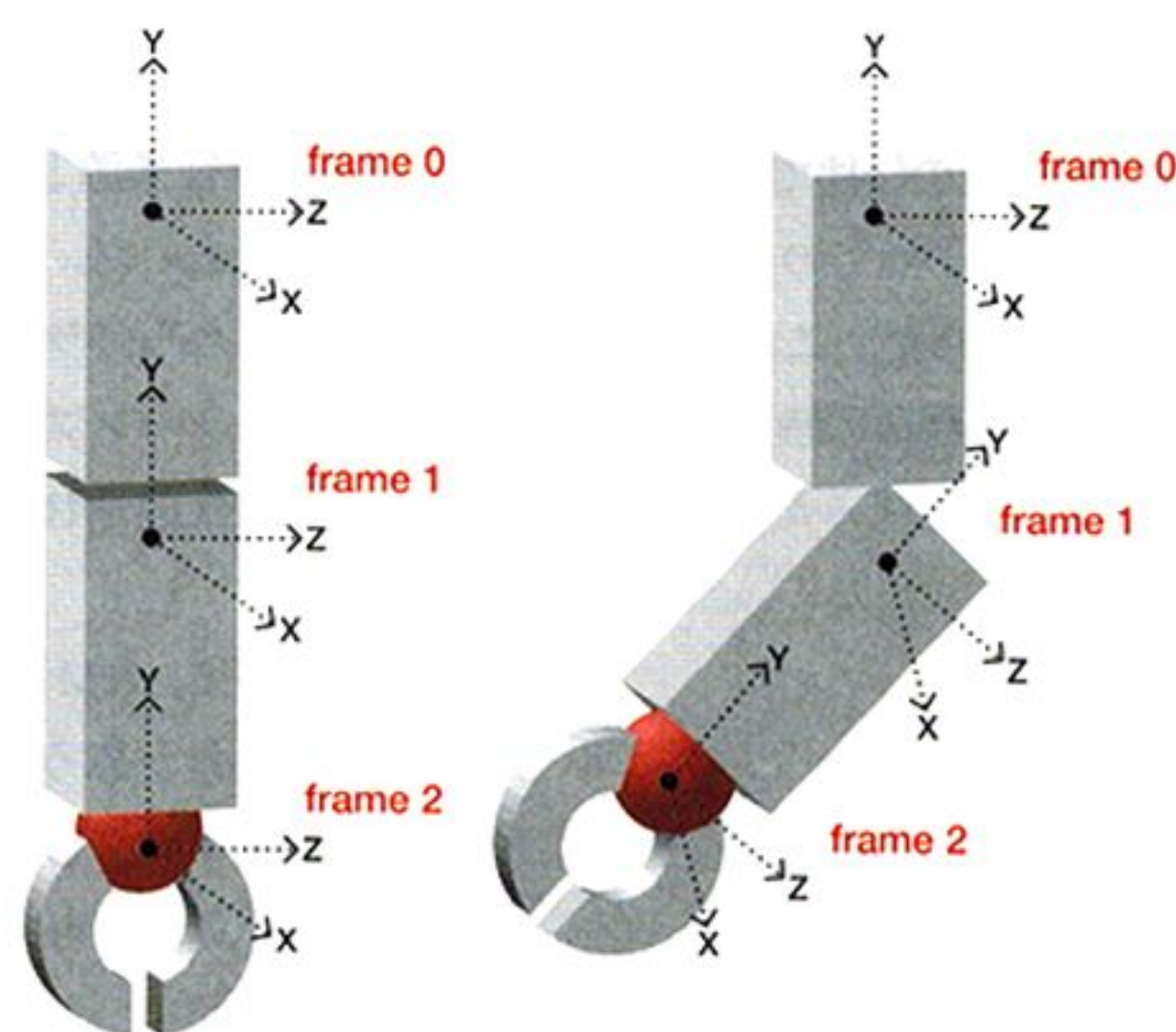
魚の動きをリアルに表現するために，少々コードが膨れてしまっているが，基本はバックサーフェスに書き込んでフリップするだけだ。あくまでも書き込むのはバックサーフェスで，プライマリサーフェスに書き込んでしまっては元も子もない。サーフェスのコピーにはBltFast()関数を用いているが，Blt()関数というものもある。こちらはBltFase()よりも遅い代わりに，拡大/縮小やクリッピングなどを行ってくれる。ただ，このクリッピングは少しでもエリア外にかかってしまうとまったく転送されないというシロモノなので（それってクリッピングっていわないと思うが），Blt()は使わずにクリッピングは自前で行っている。

かなりおおざっぱな説明になってしまったが，この先（と私の体力）を考えると懇切丁寧にというわけにはいかない。申しわけないが，引数などの詳しい説明についてはDirectXのヘルプを参照してもらいたい。

とにもかくにも，実行するとシャボン（？）の背景にたくさんのエンゼルフィッシュが泳ぎ出す。まったくウェイトを入れていないので，動きがか

図3　このような画像を用意しておく

図4　黒い部分をカラーキーにして合成表示

図5　フレームの親子関係

親フレーム（frame 1）を回転しても子フレーム（frame 2）はついてくる

なり速いのではないかと思うが，時間管理をするなり，あるいはもっと大量の魚を表示してベンチマークにするなりご自由に。魚の数はDDSample.h内のFISHNUMで定義されている（デフォルトは20匹）。

■そして3Dへ

最近のビデオカードはたいてい3Dアクセラレーション機能を持っている。したがって，Direct3Dに対応したゲームなどはバリバリ動くが，ではちょっと古めの3Dアクセラレーション機能のないビデオカードではDirect3Dは動かないかというと，決してそんなことはない。そのようなマシン環境のために，満足なスピードとはいかないかもしれないが，ソフトウェアによるエミュレーション機能も備えている。ハードウェアによる3Dアクセラレーションのためのデバイスを HAL（Hardware Abstruction Layer），ソフトウェアによるものをHEL（Hardware Emulation Layer）という。

さらにHELにはRGBモデル，Rampモデル，MMXモデルが用意されている。RGBモデルはまじめに計算しているデバイスだが，重い。それに対して，Rampモデルは光源の色は白のみだが，RGBモデルに比べて高速に動作する。MMXモデルはMMXに対応したCPUが搭載されている場合のみ使用可能で，RGBモデルの美しさとRampモデル以上の速度を備えている。

Direct3Dには2つのモードが用意されている。ひとつは基本的なインタフェイスしか用意せずに，面倒な演算などはプログラマが行わなければならないImmediateモード（IM），もうひとつは豊富なインタフェイスを備えているが，もうひとつ融通のきかないRetainedモード（RM）だ。実はRMはIMの上に載っかっており，ちょうどMFCとWin32 SDKのような関係にあるのだが，IMは想像以上に面倒なので，ここではRMで話を進めることにする。

■D3DRMでの基本概念

まずはDirect3D RMの世界観について簡単に解説しておこう。3Dである以上，3次元座標系が存在し，画面右向きがX，上向きがY，奥がZの直行座標系である。数学で習う座標系とはZ軸の向きが反対なので注意しよう。さて，座標系はこれだけしか存在しないかというとそうではなく，あくまでもこれは絶対座標である。この上にさらに座標系を載せることができ，座標軸は親座標からの位置と角度で表現される。このことを，「フレーム」という。フレームはいくらでも入れ子にすることができ，子フレームは親の座標で表されるということは，親が動くと子も伴って移動する。

こういった構造は，多関節なオブジェクトを操作するときに便利である。たとえば腕を表現するとしよう。上腕部と下碗部を別々のフレームにしてしまった場合，腕を上げようと思ったら上腕部の回転とともに下碗部も回転・移動させなければ，腕がばらばらになってしまう。これを下碗部を上腕部のフレームの子フレームとして記述してやれば，上腕部を回転させても下碗部がその位置関係を保ったままついてくる。もちろん手首から先も同じだ。ちなみに絶対座標系もフレームであり，これを特に「シーンフレーム」という。すべてのフレームはこのシーンフレームの子フレームだ。また，座標系があっても，それを見る位置を決定

しなければ画は完成しない。この「見る位置」もフレームによって表現され，これを特に「カメラフレーム」という。すべてのオブジェクトはこれらのフレームの上に載せられ，移動や回転などはこのフレームごと行われるのが一般的だ。

Direct3Dはポリゴンエンジンであって，レイトレースではない。したがって，すべてのオブジェクトは平面の組み合わせで表現される。この1枚の平面のことを，「フェース」という。さらに，このフェースを複数あわせてできたオブジェクトのことを「メッシュ」という。基本的にこのメッシュをフレームに載せて，フレームを配置していくのだが，もうひとつ足りないものがある。ライトだ。光が当たらなければ真っ暗闇である。Direct3Dでは5種類のライトがサポートされている。

●環境光源

日陰の中で直接太陽光が当たっていなくても，その中が真っ暗でなにも見えないということはないだろう。それは反射光の伝播であったり，空気中で光が乱反射して，すべての方角からわずかだが光が届いているからだ。これを環境光源という。これがなければ，物体の陰の模様などは真っ黒につぶれてしまって，まったく見ることができない。

●点光源

白熱電球などのように，1点から全方向に放射状に光を放つ光源だ。

●スポット光源

スポットライトのように，1点から扇状に光を放つ光源だ。中心の明るい部分や周囲の徐々に暗くなる部分の角度を設定することもできる。

●有向光源

並行光源のことだ。太陽は正確には点光源であるが，太陽と地球は十分に離れているため，地上では太陽光は並行光源とみなせる。このような，位置によらず光の向き（と明るさ）が一定である光源を有向光源という。

●並行点光源

メッシュ単位で並行光源となる点光源だ。点光源はその特性上，頂点ごとに光線の方向（光源までのベクトル）を計算する必要がある。頂点が少ないうちはたかが知れているが，これが何千何万という頂点数になってくると，かなりの負担となる。そこでこの並行点光源は，メッシュ単位で光線の方向を計算し，メッシュ内の頂点に関しては並行光源として処理して，実効速度を稼ぐというものだ。

これらのライトもまたフレームにくっつけられて，シーンフレーム（あるいはその他のフレーム）上に配置される。

さて，RMの世界の構造はだいたいわかってもらえたと思うが，これだけでは所詮はポリゴンのカクカクした物体が転がっているだけにすぎない。それでもしばらくは遊べそうだが，やはり現実のシーンとはほど遠い。Direct3Dはリアルタイム性を重視したゲームエンジンであるので，レイトレーシングのように美しくとはいかないが，それでもディテールアップの手法は数多く取り入れられている。

●シェーディング

陰影付けの手法である。面の向きと光源の向きから明るさを計算して，ポリゴン全体を同じ明るさで処理するのをフラットシェーディングという。それに対して，面の頂点ごとに明るさを計算し，頂点間を補間するように面の明るさを決定するのがグローシェーディングである。これにより，ポリゴンのカクカクした質感がなくなり，ぐっとディテールがアップする。さらに，ポリゴン上の1点1点ごとに光源の向きから明るさを計算する手法をフォンシェーディングという。面の向きは頂点の向きから補間される。これでさらに現実に近い効果を得られるが，計算量が多く，DirectX5ではまだサポートされていない（予約はしてある）。

●ディザ

画面で表示できる色数が少ない場合，リニアに色が変化する部分では，ピクセルごとに近似色で表現していたのでは，色の境界ができてしまい縞（マッハバンド）のように見えてしまう。これを回避するために，複数の色のピクセルを組み合わせることで中間色を表現する手法をディザリングという。24ビットカラー以上では人間の目では認

図6-1　シェーディングなし

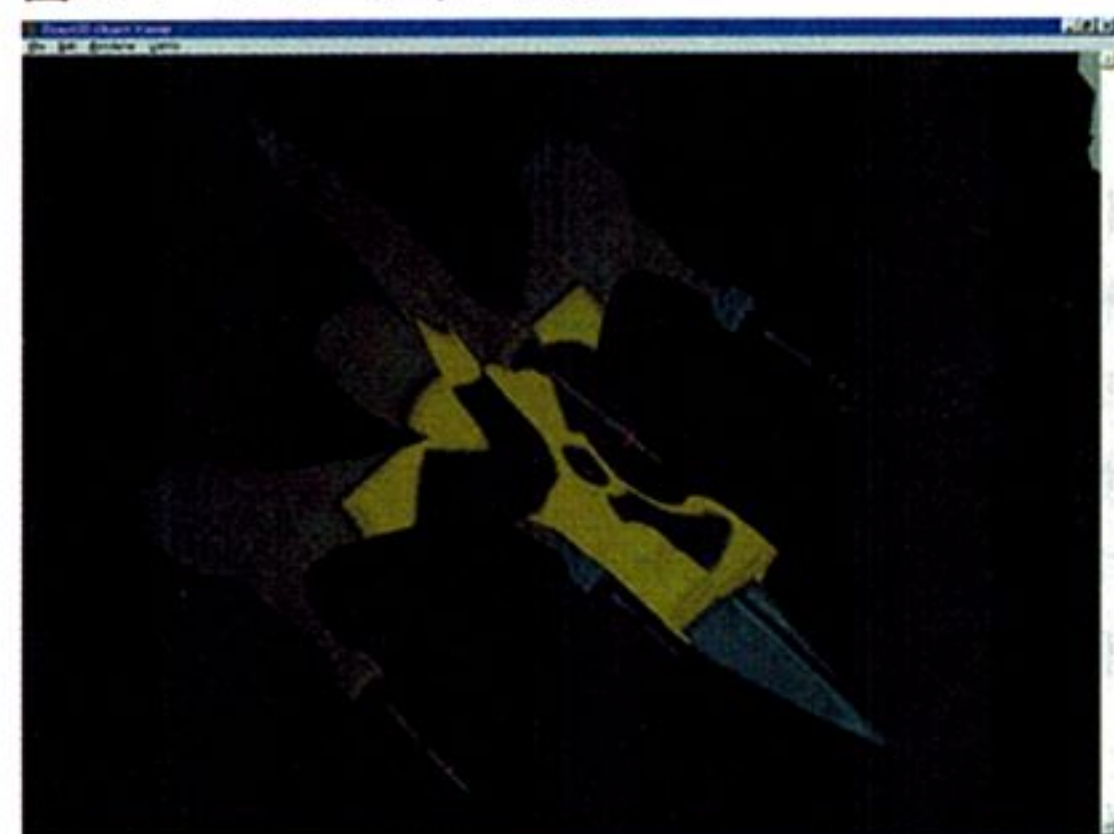



図6-2　フラットシェーディング

図6-3　グローシェーディング

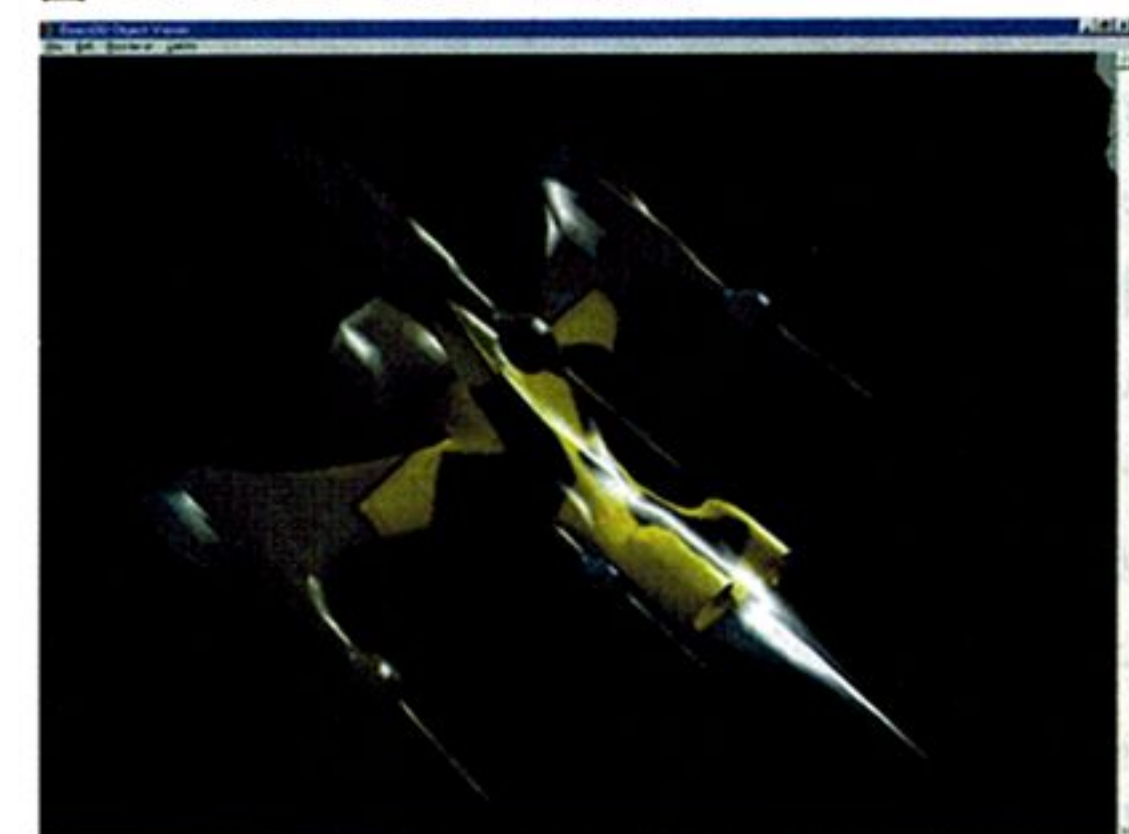



図7-1　バイリニアフィルタなしの画像

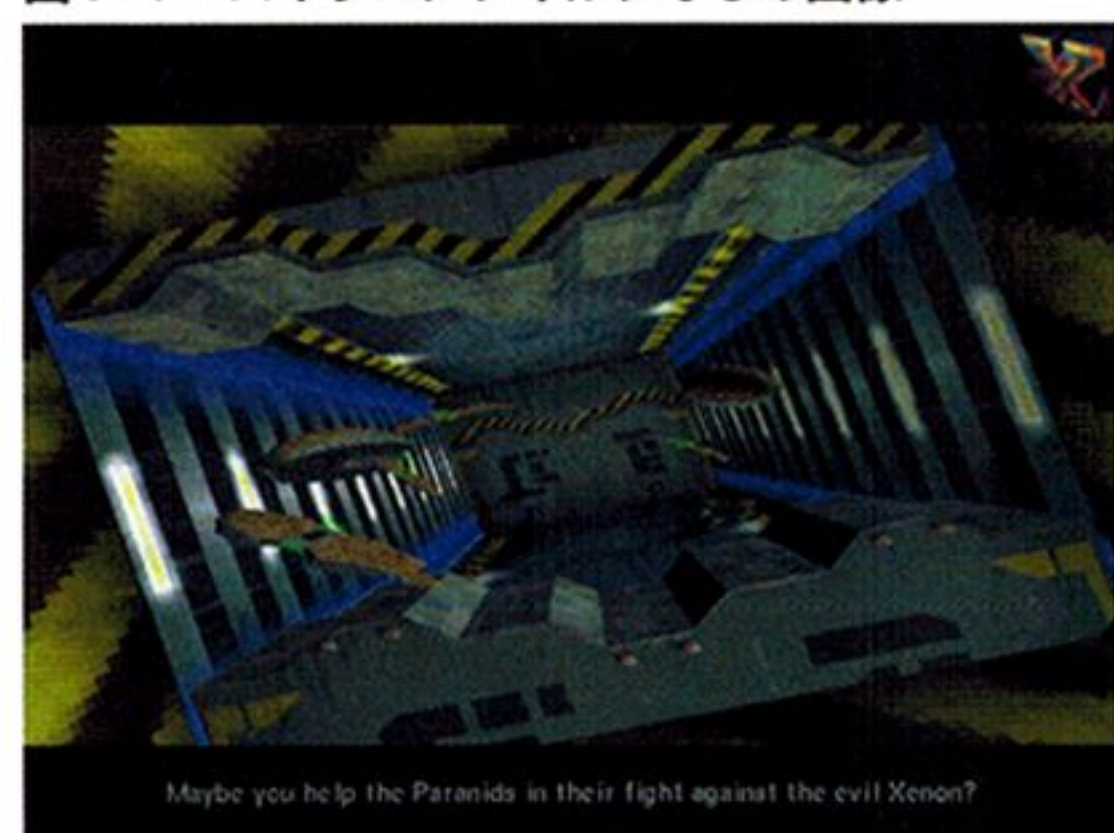



図7-2　バイリニアフィルタリングされた画像

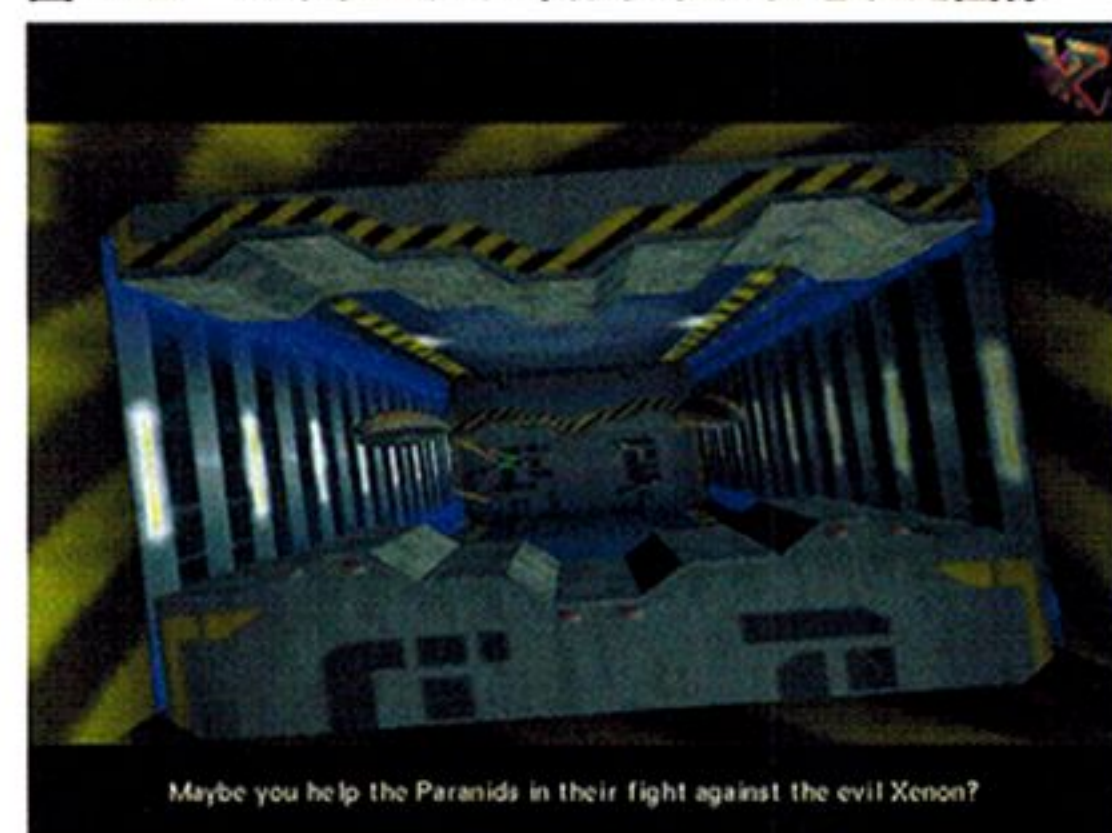



図8-1　パースペクティブコレクトされていないテクスチャ

識できないといわれているが，16ビットカラーでは色の系統によってはマッハバンドが見えてしまうので，ディザをONにすることでクオリティは上がる。

●テクスチャ

細かい模様などをポリゴンですべてまかなっていたのでは，ポリゴン数は際限なく増加し，システムがむやみに重くなる。そこで，画像をポリゴンに貼り込んでしまおうというのがテクスチャである。これにより，ポリゴンだけでは不可能に近い木目や大理石も，画像を用意して貼り込むだけで実現できる。

●バイリニアフィルタ

テクスチャは便利だが，あまり大きな画像を張り込もうとすると大量のメモリを消費する。テクスチャは(ハードウェアアクセラレーションを行う場合)システムメモリよりもビデオメモリに置いたほうが高速なのだが，ビデオメモリはシステムメモリに比べて圧倒的に小さい。そこで，テクスチャの画像はできるだけ小さくするべきなのだが，そうすると今度はポリゴンに貼り付けた場合にテクスチャのドットが四角く見えてしまう。これを回避するのがバイリニアフィルタである。このようにテクスチャドットが見えてしまうような場合，ドット間で色を補間することで，スムースに見せることができる。

●パースペクティブコレクト

テクスチャを貼る場合，単に2次元的な変形を行うだけでは奥に長いポリゴンでテクスチャに歪みが生じる。これを3次元的に行い，歪みをなくす手法をパースペクティブコレクトという。逆にいえば，テクスチャマッピングというのは単に模様を2次元変形させて貼っていただけということだ。

●アンチエイリアス

画面はピクセルドットの集まりであることはいうまでもないが，このドットがマトリクス状に並んでいるため，異なる色の境界は階段状(ジャギー)になってしまう。解像度が高ければそれほど気にもならないが，Direct3Dでは速度とメモリの関係で，VGAサイズ程度の決して高くない解像度で使われることが多い。このような場合，ジャギーは目障りな存在となるが，アンチエイリアスを使うと，ピクセル境界に比率にあわせた中間色をはさみ，ジャギーを緩和してくれる。ただこのアンチエイリアスは，現在出回っているビデオカードではあまりまじめにサポートされていないようである。

●ハイライト

たとえば金属球のようにつるつるした物体の場合，光を当てると光が反射して白く抜ける部分があるだろう。これをハイライトという。反対にざらざらしたものはハイライトは少ない。このような反射の特性など(マテリアル)をオブジェクトに与えることで，質感を持たせることができる。

●αブレンディング

ポリゴンを半透明化させて，その向こうを透過させることができる。透明化は以前からサポートされていたが，半透明化はDirectX5からサポートされた。

●ミップマップ

カメラからオブジェクトまでの距離に応じて，異なるテクスチャを貼る手法。一見して用途を思い浮かばないかもしれないが，これもディテールアップにひと役買うことができる。テクスチャを貼る場合，結果的に画面に表示されるテクスチャが元のテクスチャよりも小さくなると，そのテクスチャは間引かれて表示される。細かい描き込みをしていた場合，それが潰れて見苦しかったり，ちらついたりするのだ。そこで，あらかじめ補間縮小した画像を用意しておき，距離に応じて貼り替えてやることで，ちらつきを抑えることができる。ミップマップはこれを自動でやってくれるのだ。付録CD-ROM 2の¥DX6SDK¥SAMPLES¥MULTIMEDIA¥D3DIM¥BIN¥MIPMAP.EXEを見るのがいちばんわかりやすいだろう。

●フォグ

霧のかかったようなシーンを再現し，遠いものほど見えにくくして空気遠近法を表現できる。

ところで，こういったレンダラでよく論議されるのが陰面消去の方法だ。陰面消去とは，他のポリゴンの陰に隠れたポリゴンを画面上から消す(表示させない)ことである。現実に即せば，物体に隠れた部分は見えないのは当たり前じゃないかと思うだろうが，これがなかなか難しい。カメラから視線を飛ばし，最初にぶつかったオブジェクトを描くのが理想だが，これはレイトレースという手法で，とてもではないが(PCでは)リアルタイムレンダリングは不可能だ。

基本的にポリゴンエンジンは，3次元から2次元に変換された座標を基に次々と多角形を描画していく機能しか持っていない。ではどうするか。代表的な手法は2つある。ZソートとZバッファだ。

Zソートはポリゴンをカメラからの距離でソートしておき，奥のポリゴンから順に描いていく方法だ。奥に隠れて見えないはずのポリゴンもいったんは描画されるが，そのあとに手前のポリゴンで上書きされるために見えなくなる。これは比較的軽い手法だが，交差するポリゴンでは表示が破綻する。ポリゴン数が増えるにつれ，ポリゴンソートのコストがかかりすぎるようになるので，単純なシステム向きである。

それに対しZバッファは，画面と同じサイズのバッファを取っておき，ポリゴン描画時にピクセルごとにその距離をバッファに保存しておく。こうしておけば，次にポリゴンを描くときにそのZバッファを見て，もしこれから描くピクセルがZバッファよりも近ければ描き，遠ければ描かないことで，陰面を消去できる。この手法はZバッファ分メモリを必要とするしZソートよりも遥かに重くなるが，交差するポリゴンなども正しく描画できる。

図8-2　パースペクティブコレクトされたテクスチャ

図9　フォグの使用例。遠くが霞んで見える

図10　光の部分にαブレンディングが使用されている

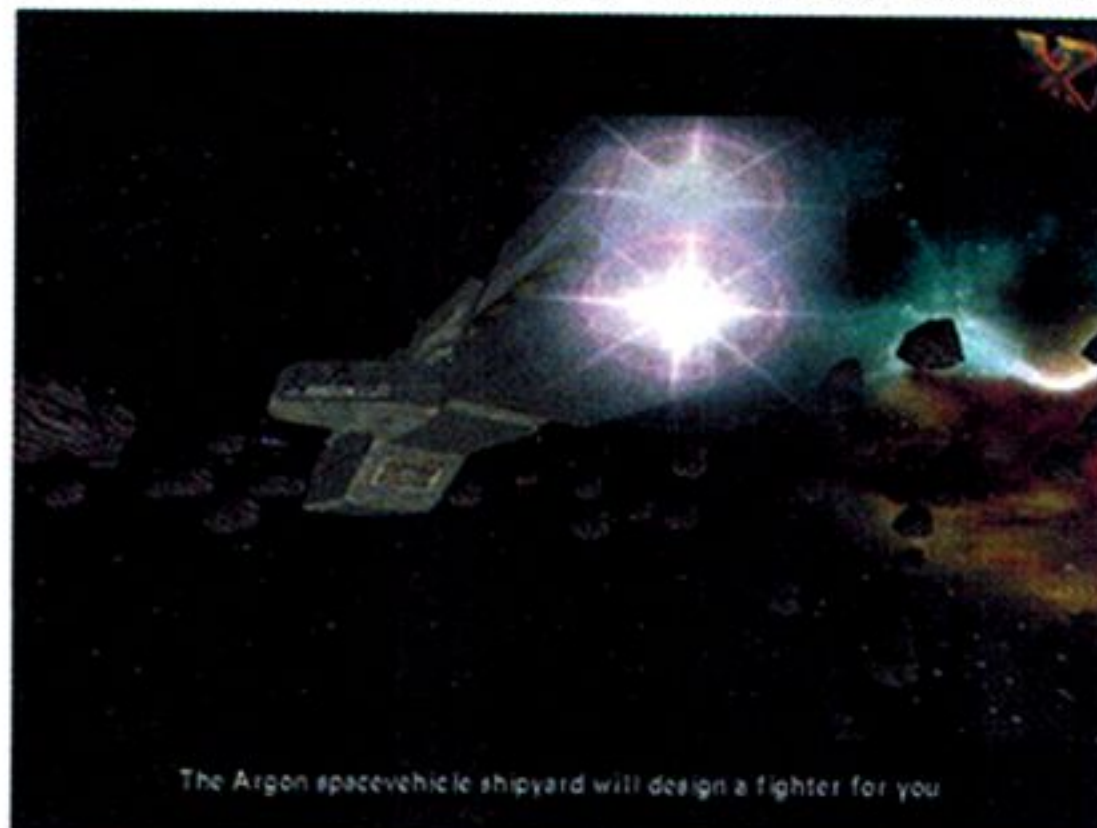



図11　鏡面反射や炎など最近流行のフィーチャー。半透明処理やテクスチャの重ね貼りなどは欠かせない

ではDirect3Dではどうかというと，一部のPowerVRなどの3Dチップなどでは独自の手法を取っているが，基本的にZバッファである。なお，このZバッファもDirectDrawのサーフェスという扱いで確保される。そしてZバッファを使わないこともできる。メモリが少ない場合や，半透明体でZバッファが破綻するのを防ぐ意味でもZソートの処理系を残しているのだろう。

とまあ，うだうだと説明するよりも，論より証拠である。こちらもサンプルを用意してあるので，見てもらいたい（D3DSample）。

■サンプルの解説
―デバイスのチェック

すでに述べたように，Direct3DはDirectDrawの上に載っかっているので，まずはDirectDrawの初期化が必要なのだが（InitDDraw()関数），先ほどのDirectDrawオブジェクトの作り方とは異なり，DirectDrawEnumerate()関数を用いている。これは，第1引数で指定したコールバック関数にDirectDrawのGUIDを次々と列挙していく関数である。なぜこのような面倒な操作が必要なのだろうか。

実は，システムにインストールされているDirectDrawドライバはひとつとは限らない。よく例に挙げられるのは，Millenniumと3D専用カードMonster3Dを搭載しているマシンの場合などだ。筆者がまさにこの環境なのだが，この場合システムにはMillenniumとMonster3Dの2つのDirectDrawドライバが存在する。

このような場合，普通にDirectDrawドライバを作成すると，プライマリデバイスであるテクスチャをサポートしていないMillenniumのDirectDrawドライバが選択されてしまう。先ほどのDDSampleはMillenniumのドライバが使用されていたことになるわけだ。2Dならばともかく，3Dをやろうというのに，Monster3Dを使わない手はない。

そこで，ドライバをひとつずつ列挙させて，そのドライバが3Dに適しているかを判断している。これはコールバック関数内での作業だ。まずは与えられたGUIDでDirectDrawオブジェクトを作成してみて，GetCaps()メソッドで能力を取得し，どのドライバを使用するかどうかを判断している。まずは3Dアクセラレーション機能をチェックし，持っていればさらにテクスチャに対応しているかを判断している。ここでMillenniumは失格となる。もし両方とも合格すれば，そのドライバを保存し，列挙を終了する。DDENUMRET_CANCELを戻り値にしているのは，もうこれ以上列挙しなくてもいいよ，という意味だ。それに対し，DDENUMRET_OKの場合は，次のドライバをちょうだい，という意味になる。勘違いしやすいので注意しよう。

とりあえずこれでMillenniumをはじくことはできるのだが，たとえばVIRGEなどの一応ひととおりの機能の揃ったカードとMonster3Dを併用している場合は，悲しいかなVIRGEが選択されてしまう（DirectX6/7ではデバイスの性能もある程度判定できるようになる）。

また，Windows 98からはマルチディスプレイがサポートされたが，セカンダリのビデオカードを使いたいといった場合も同様だ（DirectDrawだけを使う場合でも）。このようなことを考えて，本来は列挙されたドライバの名前をリストアップし，ユーザーにダイアログでドライバを選択させるというのが正統なカラー指定の方法だろう。

また，今回は640×480ドット16ビット，まずサポートしていないビデオカードはない画面モードを使用するので省略したが，そうでない場合はコールバック関数内からさらにEnumDisplayModes()メソッドを呼んで，目的とする画面モードをサポートしているかを調べる必要がある。

ところで，簡単に「合格すればそのドライバを保存する」といったが，グローバルの変数に直接保存しても構わないのだが，ここではlpContextを使うことにしよう。このlpContextはユーザーが自由に使える32ビット値で，DirectDrawEnumerate()関数の第2引数がそのまま渡される。DirectDrawEnumerate()ではDirectDrawオブジェクトへのポインタのアドレスを渡し，コールバック関数内で合格したドライバをキャストして代入しておけば，DirectDrawEnumerate()が終了した時点で適したDirectDrawドライバが(もしあれば)lpDDに格納されているというわけだ。

さて，この時点で，Millenniumしか搭載していなかったり，3Dに対応したビデオカードを搭載していない場合は，まだDirectDrawドライバは作成されていない。その場合はしかたがないので通常の方法でDirecrDrawドライバを作成し，HELを使うことになるのだが，HELについてはまたあとの話だ。協調レベルと画面モードの設定は先ほどと同じだが，今度は画面モードを16ビットカラーにしてある。

プライマリサーフェスとバックサーフェスの作成（CreateOnScreenSurface()関数）も，同じようなものだが，違う点はプライマリサーフェスの作成時にCapsにDDSCAPS_3DDEVICEを指定すること。いうまでもないが，3D機能を使うよ，という宣言だ。あとは，念のためにBlt()メソッドでサーフェスを初期化している。これは，Monster3Dなどの3Dカードでは，以前のビデオメモリの状態が残っている場合があるからだ。

■もろもろの初期化

この先がDirect3D部分である（InitD3D()関数）。Direct3DオブジェクトはDirect3DCreate()とかいう関数で作ったりするわけではなく，IDirectDrawのQueryInterface()メソッドを使って取得する。このQueryInterface()とは何者なのだろう。

インタフェイスというのは，関数のアドレスの束である。IDirectDrawインタフェイスにはCreateSurface()とかGetCaps()といったメソッドがあるが，これはあくまでも表面上のこと，その実体はビデオドライバそのものだ。ビデオドライバを2Dの側面から見た場合にはIDirectDrawインタフェイス，3Dの側面から見ればIDirect3Dインタフェイスとなり，実体は同じなわけだ。

しかも，Direct3DはDirectDrawの上に載っているわけだから，Direct3Dオブジェクトを取得しようと思ったら，必ず，すでにDirectDrawのオブジェクトが作られているはずである。QueryInterface()は，このような「別の側面から見たインタフェイスをちょうだい」というメソッドなのである。

第1引数のIID_IDirect3D2で，IDirect3D2のインタフェイスを要求しているわけだ。IDirect3D "2" というのは，DirectX5で拡張されたDirect3Dインタフェイスだ。どうせ使うなら新しいほうがよい（もちろんマシンにDirectX5以上がインストールされていないとエラーとなる）。いまいちしっくりこないかもしれないが，いまはこういうものだと思っておけばとりあえず問題はないだろう。

次はデバイスの取得だ。これにはまたコールバック関数による列挙を行う。ここで列挙されるデバイスはRGBやRamp，HALといったものだ。本来はカラーモデルや性能を評価するべきなのだろうが，ここではHALがあればそれでOKとしてGUIDを保存している。さてGUIDが取得できたら，デバイスを作成する前にZバッファを作成しておこう。

Zバッファもサーフェスであると述べたが，基

本的にCapsにDDSCAPS_ZBUFFERを指定するだけだ。注意が必要なのは，HALの場合はZバッファはビデオメモリ内に取る必要があるということだ。HELの場合はどちらでも構わないが，ビデオメモリはシステムメモリほど大きくはないので，システムメモリにしておくほうが無難だ(速度的にも有利だろう)。

また，Zバッファのビット深度は16ビットとした。ビデオカードによっては24ビットや32ビットをサポートしているものもあり，デバイスのコールバック関数中でlpHWDesc->dwDeviceZBufferBitDepthでサポートしている深度を取得することができるが，16ビットをサポートしていないというのも考えられないので，またしても省略させてもらった。

Zバッファが作成できたら，バックバッファにアタッチする。ここで，プライマリではなくバックバッファにアタッチするのは，あくまでも書き込みはバックバッファに対して行うからである。

そのあと，そのバックバッファを指定して，デバイスを取得する。HALが対応していれば先ほど保存したGUIDで作成，そうでなければまずはMMXで作成してみて，だめならば(CPUがMMXに対応していなければ)Rampデバイスにしている。より正確に表示したいのならば，ここをRGBデバイスに変更するとよいだろう。また，Millenniumしか搭載していないマシンでは，MilleniumのHALが使用され，このままではテクスチャが表示できない(エラーになるわけではないが)ので，各自でテクスチャをサポートしていない場合はHELにするなどの改良を加えてほしい。

■RMの関数を使う

いよいよDirect3D RMの出番である(InitD3DRM()関数)。まずはDirect3DRMCreate()でDirect3D RMオブジェクトを生成したら，先ほどのQueryInterface()でIDirect3DRM2インタフェイスを取得する。これもDirectX5で新しく追加されたインタフェイスである。さて，ここで用済みのIDirect3DRMインタフェイスはさっさと解放している。いままで説明しなかったが，RELEASE(lpOldD3DRM)というのはオブジェクトを解放するマクロで，lpOldD3DRM->Release()に展開される。いらないものは解放する，というのは当たり前のようだが，ちょっと待て，QueryInterface()で取得したlpD3DRMはlpOldD3DRMと実体は同じものだから，解放しちゃまずいんじゃないの？ と思うかもしれない。しかしご安心あれ。解放されるのはあくまでもインタフェイスだけだ。もう少しいえば，オブジェクト自体は参照カウンタによって管理されており，そのカウンタがゼロにならなければ解放されることはない。

参照カウンタはDirect3DRMCreate()でももちろんカウントされ，QueryInterface()した時点でさらに加算される。つまりlpOldD3DRMをリリースして参照カウンタがデクリメントされても，QueryInterface()で加算された分が残っているので，解放はされないのだ。

あとは流れ作業だ。Direct3D RMデバイスを取得し，シーンフレーム，カメラフレーム，ビューポートを作成し，バッククリップの設定をする。シーンフレームというのは前にもいったとおり，絶対座標系を持ったフレームで，すべてのフレームの親である。したがって，CreateFrame()メソッドの親フレームを指定する第1引数をNULLにしてフレームを作成する。カメラフレームは，視点となるフレームである。通常のフレームと同様に作成するが，ビューポートを作成するときに，そのフレームを指定することで，カメラフレームとなる。

バッククリップというのは，奥行きのクリッピング(一般的にはFarClipという)である。一定以上遠くにあるものは描画しないようにするためのものだ。適当な値を放り込んでいるが，今回はあまり意味をなしていない。なお，D3DVAL()というのはマクロで，Direct3Dで数値を指定する場合には必ず指定するお約束だ。

ここまでは，Direct3D RMを使ううえでの常套手段みたいなものだ。スケルトンとして使っても構わない。ここからがやっとアプリケーション固有のオブジェクトの配置である(BuildScene()関数)。まずはクオリティの設定から。陰を32段階，テクスチャで使用される色数を256色(画面上での色数のことであって，テクスチャとして指定できる画像ファイルの色数のことではない)，ディザをON，テクスチャにはバイリニアフィルタをかける。光源は点光源と環境光源の2つ，作成した光源フレームに取り付ける。点光源と環境光源は方向がないので，フレームの原点だけが問題

図13　立方体に絵を貼る。基本中の基本だ

図12　立方体の法線とuv座標

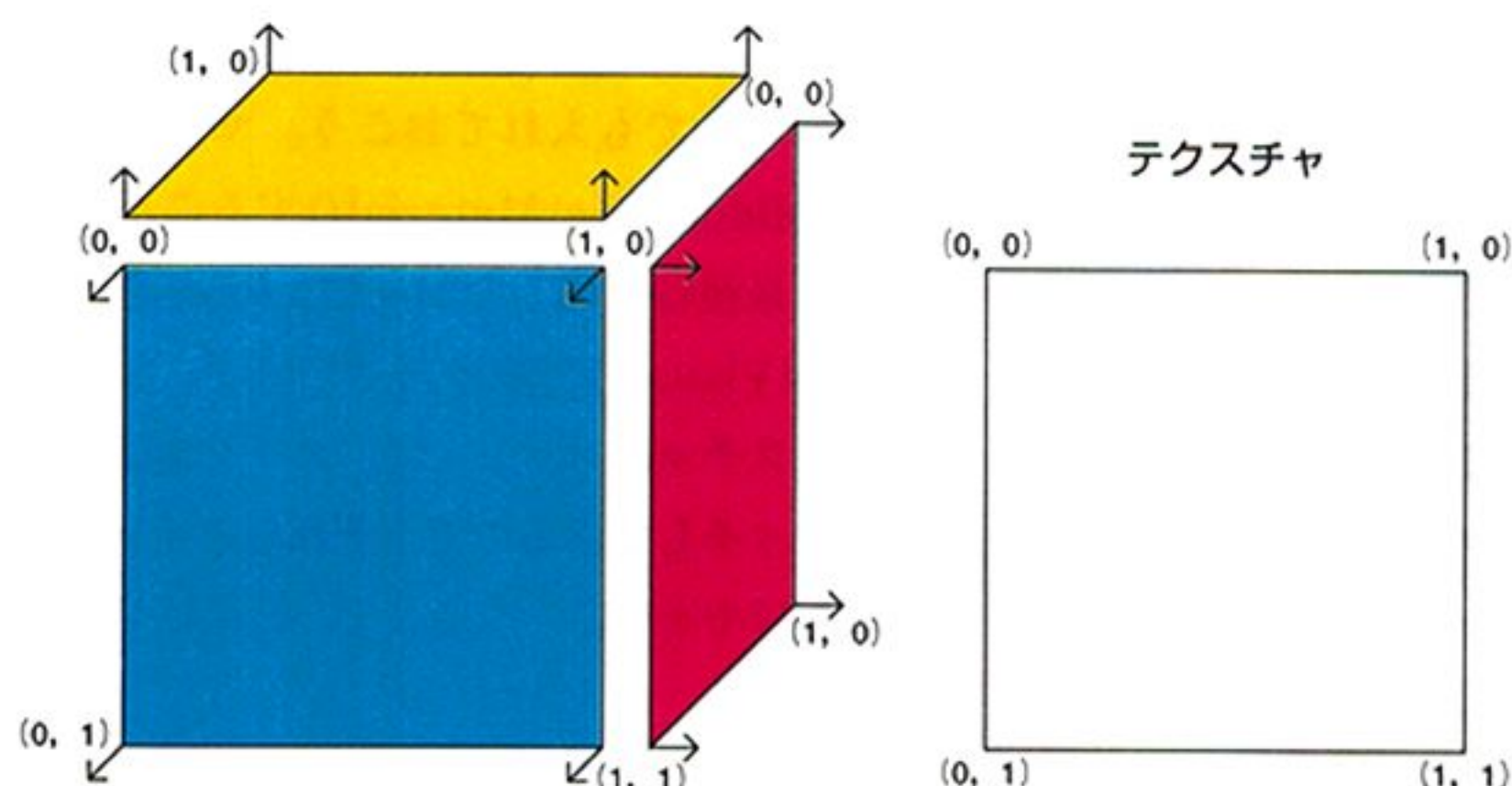



図14　生成した地形上を遊覧飛行してみる

図15　ヘリとカメラの関係

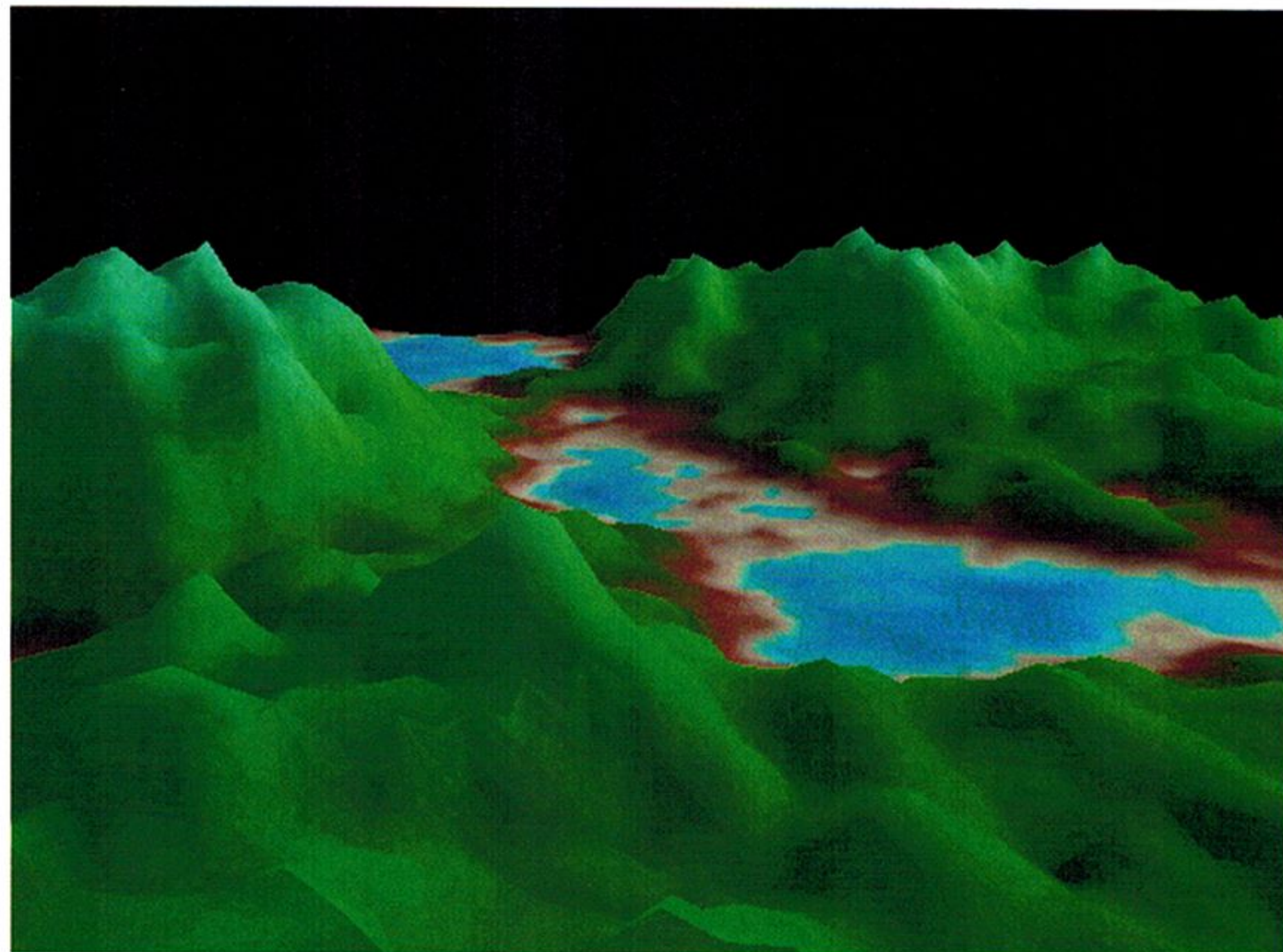
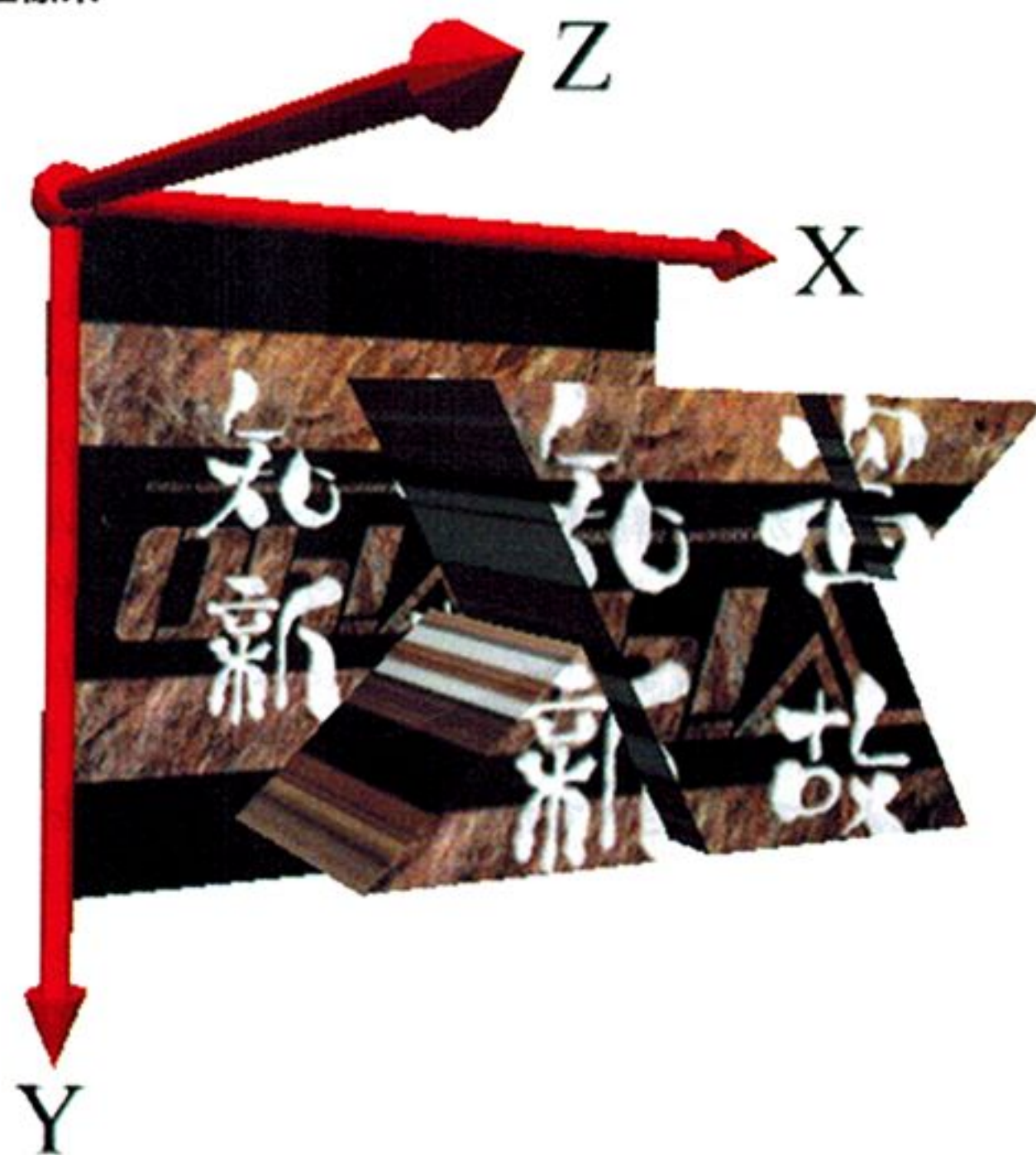
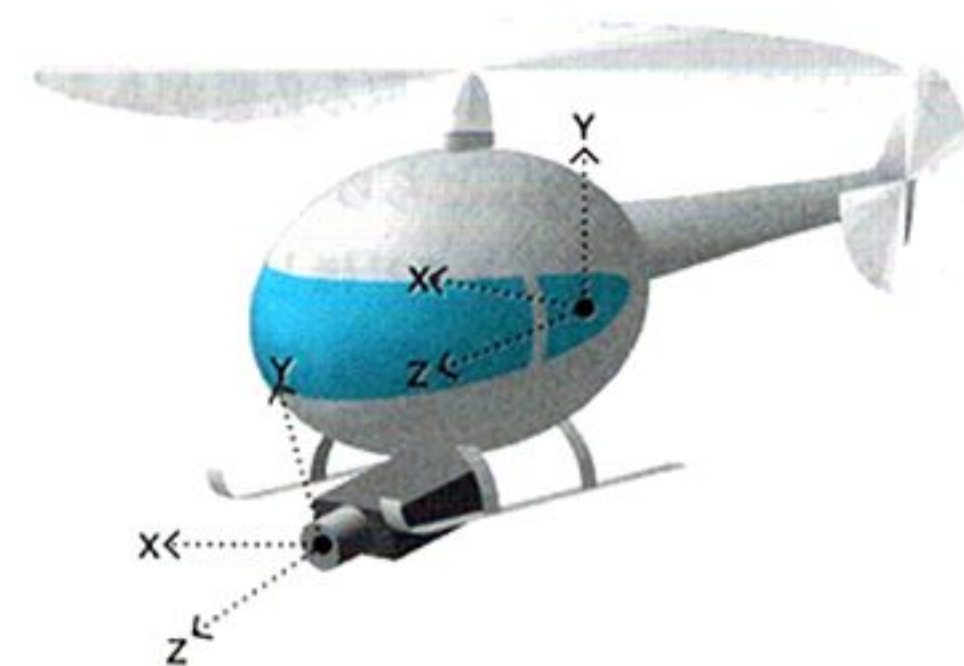



図16　投影マッピングの座標系

だが(環境光源は原点も関係ない)，有向光源などはフレームのZ軸の向きが光の向きとなる。

ここで，AddLight()でフレームにライトを取り付けたあとでpLightをリリースしているのは，先ほども話したようにリリースしているのはあくまでもインタフェイスだけであって，AddLight()した時点で参照カウンタはインクリメントされているからである。ひとつのフレームに2つのライトをくっつけているが，ライトだけでなく，フレームには複数のメッシュも混在させてつけることができ，カメラフレームにライトをつければ，自分が辺りを照らしているような効果を得られる。

■メッシュの作成

ひととおり環境が揃ったので，あとはオブジェクトだ。メッシュを扱えるものにはMeshとMeshBuilderがあるが，Meshは基本的なメソッドしか装備されておらず，少々扱いづらいのでMeshBuilderを使うことにしよう。

CreateMeshBuilder()でMeshBuilderを作成したら，あらかじめ配列に用意されていたフェースデータをAddFaces()メソッドでMeshBuilderに追加している。このデータは1辺の長さが1の立方体データだが，注意が必要なのは法線ベクトルだ。立方体のひとつの頂点は3つの面の頂点を共有しており，この3つの頂点の法線をまとめて設定することで，グローシェーディング時に面がスムースに結合される。

ただ，現在の場合はシェーディングをかけたくないので，面の頂点それぞれに法線を与えており，合計6×4=24の法線を与えている。念のためにSetColorRGB()で色を白に，ちょっと小さいのでScale()で3倍くらいに大きくしておこう。これでキューブのメッシュは完成だ。

引き続いてそのキューブにテクスチャを貼ってみる。テクスチャはあらかじめ用意した画像を貼るのだが，ファイルを開いて，ヘッダを解析して……なんて面倒なことは必要ない。なんとファイル名を指定してLoadTexture()一発で簡単に読み込めてしまう。指定できるファイルがBMPとPPMだけというのがナンだが，BMPはRLE圧縮にも対応しているので，まあよしとしよう。

ただし，テクスチャサイズは640×480か，あるいは2の累乗という制限がある(64×128とか)。こいつをMeshBuilderのSetTexture()でセットするだけなのだが，これだけではメッシュのどこにどのようにマップするのかわからない。そこで，頂点ごとにテクスチャのuv座標を指定するといった面倒な作業を行う必要がある。今回はキューブを頂点8つで作っており，適当にuv座標を指定しているので，テクスチャが裏返ったりしているが，各面にきっちりテクスチャを貼ろうと思えば，法線同様，面ごとに頂点を指定する必要があるだろう。忘れてはならないのがパースペクティブコレクトだ。SetPerspective()で設定しないと，パースのかかった面でテクスチャが歪んでしまう。

ここまででテクスチャが貼られたキューブは完成なのだが，一応マテリアルを設定して，適当にハイライトでも入れておこう。マテリアルもMeshBuilderでSetMaterial()することで設定できる。最後は完成したMeshBuilderをフレームにAddVisual()で取り付けて完了。マテリアルとテクスチャはメッシュに，メッシュはフレームにアタッチしてあるので，それらのインタフェイスはもはや不要ということで，最後にリリースしている。

空間のデザインができたので，レンダリングし

てみよう(Render()関数)。まずは以前のビューをクリアしておき，レンダリングしてプライマリとバックバッファをフリップする。ここで，lpD3DRMView->Render(lpD3DRMScene)はあくまでもlpD3DRMScene(つまりシーンフレーム)以下のフレームを描画するようにビューに準備をさせるというだけで，実際に描画が行われるのはlpD3DRMDev->Update()である(と思う)。

初期化やメッシュ作成での手間を考えると，レンダリングはあまりにもあっけない。ただ，これだけでは最初に配置したシーンを表示するだけなので，カーソルキーでキューブが回転するようにしてある。メッセージポンプの中だ。あえてキーダウンメッセージなどではなくGetAsyncKeyState()を使ったのは，メッセージではシステムで設定されているキーリピート間隔などが考慮されてしまうからだ。GetAsyncKeyState()ならばリアルタイムでキーの状態を取得できる(どうせならDirectInputを使えよという意見もあるだろうが)。カーソルキーを押された場合は，キューブのフレームをAddRotation()で適当に回転させ，Render()でレンダリングさせている。

ここで，AddRotation()の第1引数だが，ヘルプには「新しい回転を現在のフレーム変換と合成する方法を示す」となっている。なにをいっているのかいまいちピンとこないかもしれないが，要はこういうことだ。すでにフレームになんらかの回転(あるいは移動)が加えられている場合，いまから指定する回転をその回転(あるいは移動)の前に行うかあとに行うか(あるいは置き換えるか)を示している。ますますなんのことやらわからない？ じゃあ，D3DRMCOMBINE_AFTERをD3DRMCOMBINE_BEFOREにしてコンパイルしてみるといい。それがいちばんわかりやすいだろう。

いかがだろう。とりあえずテクスチャの貼られたキューブが画面に表示されるはずだ。カーソルキーを押すと，上下左右にくるくる回る。いくら待っても表示されない場合は，カーソルキーを押してみよう。マシンによっては最初のRender()関数で正常にレンダリングできない場合があるようだ。

■Xファイルと投影マッピング

さてさて，キューブを表示できて感動もひとしおかもしれないが，たかだかキューブ1個表示させるのにこんなに面倒なこと……とも思っているかもしれない。初期化は使い回しができるからまだいいとして，やはり面倒なのはメッシュのフェースデータとテクスチャコーディネートだろう。立方体程度ならこのくらいで済むが，面が増えてくると爆発的にデータが増え，有機的な形状のオブジェクトなんて気が遠くなる。実際問題として，フェースデータを今回のようにあらかじめ用意した配列で渡すケースはあまりない(少なくとも手で書くことはまずない)。

図17　環境マッピングの例。貝の内側は描画されていない

ではどうするか。DirectXではオブジェクトデータファイル形式として，Xファイルというものを用意している。このXファイルの中にフェースデータを構築しておいて，プログラムで読み込もうというわけだ。オブジェクトは市販のモデラなどを使って作成すればよい。直接Xファイルに出力できるモデラは多くはないが，オンラインソフトなどでXファイルにコンバートするツールはそこそこ存在する。また，こういったコンバータがテクスチャに対応していない場合でも，Direct3D RMでは「ラップマッピング」という手法で，いちいち頂点ごとにuv座標を指定しなくてもある程度のマッピングはできるようになっている(内部的にはフェースごとのuv座標に展開される)。次はそのXファイルと投影マッピングを利用してみよう。

使用するオブジェクトは，ご存じの方も多いかもしれないが，筆者が昔作った地形生成プログラムで作成されたデータをXファイルにコンバートしたものだ。この地形にテクスチャマッピングを施し，その上空を遊覧飛行してみよう。初期化部分はほとんど変わらないが，カメラフレームに少々変更を加えている。たとえばヘリコプターから地上を撮影する場合，移動はヘリの座標系となるが，カメラ自体は水平よりも下を向けるだろう。

つまり，正確にはlpD3DRMCameraはヘリのフレームであり，ローカル変数のcameraが本当のカメラのフレームだ。cameraはlpD3DRMCameraに固定してしまうので，すぐに解放しているが，このフレームもグローバルで宣言すれば，あとからカメラを遠隔操作することもできる。光源などは先ほどと同じだが，今度はグローシェーディングをかけてみるので，IDirect3DRMDeviceのSetQuality()メソッドで設定しておく。

さて，メッシュだが，これもMeshBuilderを使う。Xファイルからメッシュを読み込むには，Load()メソッドによって，これまた一発で読み込めてしまうのだ。これでおしまい。なんとも拍子抜けである。

なお，デバイスでも指定したが，メッシュ単位でもSetQuality()メソッドでグローシェーディングをするような指定が必要だ。さて，むしろ複雑なのはマッピングのほうだ。テクスチャの読み込みは先ほどと同じだが，これをラップマッピングするには，IDirect3DRMWrapというインタフェイスを使う。ラップっていうのは，料理に使うラップなんかと同じで，オブジェクトをどのようにテクスチャで包み込むかを指定するものだ。Direct3D RMでは，平面マッピング，円柱マッピング，球体マッピングをサポートしている。

もうひとつ環境マッピングもあるのだが，これはほかとはちょっと性質が違うものなので，またあとで説明しよう。平面マッピングとは，テクスチャを特定の方向から平行に投影したようにマッピングする方法だ。今回はこの方法により，地形の上からテクスチャを投影する。円筒マッピングとは，メッシュ全体を円筒で包み込み，円筒の側面から軸に向ってテクスチャを投影する方法だ。さらに球面マッピングはテクスチャをメルカ

トル図法的に投影する。

これらのラップが複雑だといったのは，ラップを作成するCreateWrap()関数の引数が少々多いからだ。たとえば平面ラップでも，メッシュに対してどの方向から，どの角度・位置・サイズといった情報が必要となる。つまり，ラップにも座標系があるのだが，ラップはフレームを使ったりせずに，CreateWrap()の引数でその座標系をすべて指定することになる。座標軸に対してのラップの方向は図のとおりだ。また，第14・15引数で指定する縮尺係数は，長さ1の間に何枚のテクスチャを貼るかを示す係数なので，小さいほど大きく拡大されてテクスチャが貼られることになる。

wrapはbuilderにアタッチしてしまったら，例によって不要であるのでリリースして構わない。あとは，カーソルキーが入力されたときに，それに従って今度はカメラフレームを移動させたり回転するように変更するだけだ。

空中散歩は楽しめただろうか。地面と湖面の法線が独立しているので境界が出てしまうが，それでも結構雰囲気が楽しめるのではないかと思う。あと，湖岸線がちらつくことがあるが，これはZバッファの精度の問題だと思われる。Permediaではちらつかなかったので，同じ16ビットのZバッファでもビデオチップによって演算精度は異なるようだ。

ちなみにこの地形は2次元FFTによって作成された周期関数であるので，複数並べると四方がぴたりとくっつく。地平線になるくらいまで並べれば，それはそれは雄大な景色になりそうだが，あまりやりすぎるとむやみに重くなるのでほどほどに。

ところで，今回はメッシュの入ったXファイルを用いたが，Xファイルで保存できるのはメッシュだけではない。フレームやアニメーションも保存しておき，必要に応じてロードすることができる。

■環境マッピング

さて，最後に先ほどあと回しにした環境マッピングに手をつけておこう。環境マッピングはクロムマッピングともいわれ，周囲が映り込んだ金属のような質感を与えることができる。とはいっても，実際に映り込みを計算すると膨大な演算量になるため(なのだが，実際に映り込みをリアルタイムレンダリングしてくれるチップが現れるのもそう遠い将来ではなさそうだ。恐ろしい時代である)，あくまでもマッピングにより「っぽく見せる」だけである。これには工夫が必要だ。というのも，見る位置や角度によってテクスチャを移動させる必要があるからだ。

そこで，オブジェクトが載ったフレームが動くたびに，ラップを設定し直すようにする。具体的にはIDirect3DRMFrameのAddMoveCallback()関数を使う。この関数は，フレームが動くたびに指定したコールバック関数を呼んでくれるので，この中でラップを設定すればいい。ところで，いままではCreateWrap()の第2引数であるラップのための参照フレームは，オブジェクトが載ったフレームを指定していたのだが，今度はカメラフレームを指定している。これは，あくまでも環境マッピングが視点を基準に設定する必要があるからだ。このようにカメラフレームを基準にラップを作ってしまえば，あとはコールバック関数中でIDirect3DRMWrapのApplyRelative()を呼ぶだけで自動的にカメラフレームとオブジェクトの位置関係からテクスチャコーディネートを更新してくれる。

あと，マッピングの性質上，メッシュが動かないとあまりうれしくないので，話が前後するが，フレームはSetRotation()で自動的に回転するようにしてある。さらに，これだけではちょっと寂しいので，メッシュをもうひとつ置いておこう。環境マッピングされた「Xロゴ」の周囲を，唐突に「オウム貝」が自転しながら公転する。こういった動きを実現するには，フレームの親子関係を利用するのが便利だ。まず回転するフレームをひとつ用意する。これが公転運動となり，このフレームの原点が公転の中心となる。さらにこのフレームの子フレームを作り，原点を適当な距離だけ移動させる。この距離が公転半径だ。さらにこの子フレームを回転させる。これが自転となる。したがって，この子フレームにメッシュをくっつければいいわけだ。

ところで，SetRotation()で「自動的に回転する」とはどういうことだろうか。SetRotation()が呼ばれた段階ではまだフレームに回転は加えられていない。指定した回転は，IDirect3DFrameのMove()メソッドが呼ばれた時点ではじめて加えられるのだ。しかも，Move()メソッドを呼んだら呼んだ回数だけ加えられる。慣性みたいなもんだ。このMove()は特定のフレームの子フレームすべてにわたって有効であるので，シーンフレームでMove()すれば，SetRotation()で回転属性を与えられたフレームすべてが回転することになる。

さらにいえば，Move()は引数で回転量を指定することができる。つまり，SetRotation()で指定した回転とMove()で指定した回転量を掛け合わせた分だけフレームは回転する。つまり，レンダリング時間間隔を計測し，それに比例した値をMove()に渡してやることで，マシンのパワーに関わりなく一定の速度でフレームを回転させることができるのだ。なお，Move()は回転だけでなく，SetVelocity()関数による速度属性にも有効だ。

動かしてみると，Xロゴに「温故知新」が映り込んでいるようには見えるが，当然のことながら周囲を回るオウム貝が映り込むことはない。オウム貝までも映り込んで見せるには，2段階のレンダリングという技が必要だが，今回はここまでということにしておこう。

また，よく見るとオウム貝の中が正常にレンダリングされていないことにも気づくだろう。多くのポリゴンエンジンがそうであるように，Direct3Dのフェースにも表と裏がある。頂点が時計回りに指定されているほうが表だ。フェースは表から見ると，正常に1枚の板に見えるが，裏からは見えない(素通しになる)。したがって，両方から見える板を作ろうと思ったら，2枚のフェースを張りあわせなければいけないのだが，オーム貝は外側にしかフェースを張っていない。これも以前に作ったデータの流用のため，こういうことになってしまっているが，必要ならば各自でなんとかしてもらいたい。そうそう，Direct3Dに対応してないビデオカードならZバッファとかテクスチャはシステムメモリに作られるから大丈夫だと思うが，HALがあるがビデオメモリが2MBしかないビデオカードでは，ビデオメモリが足りなくて正常動作しない場合があるかもしれない。そんな場合は画面モードを320×240に変更するなどして対処してもらいたい。

あと，すっかり忘れていたが，Direct3Dを使ってプログラムを書くときには，DirectDrawに加えてヘッダにd3drm.h，ライブラリにd3drm.lib，dxguid.libを加える必要がある。

●今回はここまで

ああ，ごめん……。実は私もDirect3Dは始めて2～3カ月ほどだったりする。DirectDrawのほうはずいぶん前にちょっといじったんだけど，それもずっとほったらかしっぱなし。結構前から(U)氏にはせっつかれてたんだけどね，やっぱせっぱ詰まらないとなかなかねぇ。本当は今回ちゃんとしたライブラリみたいなものを作っておきたかったんだけど，残念ながら間にあわず。もっと数学的な説明も(読者が望んでいるかどうかは別として)したかったんだけど，学生やめてずいぶん経つもんで，そういう頭もすでに持ちあわせていない。結局フィーリングだけで押し通してしまったんだけど，わかってもらえたかねぇ。ちょっと不安。次回までにはウィンドウモードとかも含めたライブラリを作っておくので，それまでに今回のところまでは理解しておいてもらえるとうれしい。というわけで，次回は2段レンダリングとかアニメーションとかやるぞ～！　ああ，勉強しとかなきゃ。

シーラの3Dプログラミングのススメ

シーラ/マイクロネット

3D表示でなにかを作りたい。と思ってみてもそう簡単にはいかない。敷居が高すぎる。Direct3Dが出現してずいぶん敷居が低くなった。しかしイニシャライズの大変さや，VC++による開発など敷居が高いことに変わりはない。マイクロネット開発部隊提供によるVBでDirect3D RMを扱うDLL。これを使って3Dプログラミングに挑戦しよう

■シーラって誰ですか？

いきなり「シーラ」などといわれても知ったことであるはずがない。別に私は外人っぽさを気取るつもりはないのである。シーラとはシーラカンスのことである。私はいまだ現役のゲームプログラマで，最近はずっと3Dアトリエ開発に携わっている。

この仕事を始めてから15年を経過してしまった。業界内でもかなり高齢に属しているという自信があったりする。よって私はシーラなのである。

ここ数年でゲームの表示形態はすっかり3次元にさま変わりしてしまった。2次元表示であった頃は，算数がわかればゲームのプログラミングができたものである（企画力やデザインセンスは別の話）。しかし3次元を扱うとなるとどうしても「数学」という外国語の学力が必要になってしまう。

もう4年ほど前になってしまったが，SATURNとPlayStationの発売とともにこの状況は始まった。当時は3次元ゲームを自作しようとしても自分のパソコン上で動かすのは処理速度の面から，ほとんど不可能だった。

3Dが主流になって以来，ハード面，ソフト面両方から一般の人々にとってゲーム制作は遠いものとなってしまっていたのである。

ところが最近になって，DOS/Vマシンの高性能化，3Dビデオカードの普及，DirectXの発表に伴い，環境は見違えるほどよくなった。自分が普段使っているパソコンで3Dゲームが開発できるようになったのである。しかしながら，プログラミングが高度になり，なかなか手を出せないという問題が残ったのである。

今回，Oh!Xから機会をいただきVBからDirectX RMをコントロールするDLLを発表することになった。

■なんのためのDLLなのか

ゲームの制作現場では，ゲーム開発用のツールが山ほど必要になる。ハードメーカーからもある程度は提供されるが，値段が高価だったり，機能的に不満だったり，待っている時間がないなどの理由でなかなか「これ」というものは手に入らない。究極の選択としてはやはり自分で作るしかないのが実情である。

プログラミングするうえで，VBというのはなかなかどうして侮れない言語である。ツールのよし悪しのほとんどを占めるユーザーインタフェイス部分を作るのにはとても威力を発揮する。ということで，我社ではユーザーインタフェイス部分とスピードのいらない部分はVBで制作し，スピードの要求される部分はVC++（といってもコードはほとんどC言語）でDLLを制作し，表示はDirect3Dに任せるという方法が主流になっている。

今回発表させていただくDLLはDirect3D RMを使用している。ツール制作に特化されてはいるが，ひととおりの機能を備えているので，作りようによっては簡単なゲームの制作だって可能である。興味のある方や超初心者の方は，マイクロネットホームページ「サルのゲーム製作講座」，

http://micronetclub.co.jp/

を覗いていただきたい。

■シーンの構成

カメラ1，平行光源2，読み込むオブジェクトはX形式か，3DA形式，回転，移動，拡大縮小が可能で，親子関係をサポート。その他もろもろというところが仕様である。図1に世界の略図を示す。座標系は3Dアトリエ座標系（SATURNライブラリ，Softimage，PRISMSなどと同じで，PlayStationライブラリ，3D Studio MAXとはZの符号が逆）を用いている。なぜそうなのかというと，3Dアトリエを開発した当時我社のデザイナーたちが慣れていたSoftimageやPRISMSと同じにしたというだけのことである。

■サンプルプログラムについて

サンプルのEXEファイルと，そのVBのソースコードがCD-ROMに付属されている。ある程度VBがわかっている人であれば，ソースを見ていただければだいたいの使用法はわかってしまうはずである。

機能的には，Xおよび3DA形式の3次元データのビュアである。データのロードや表示については，ソースを追いかけていけば具体的な方法の見当がつくはずだし，このソフト内で使われていない機能に関して，追加などしていろいろ実験し理解していくための材料と考えてほしい。

といいながらもそれで終わってしまってはあまりにも冷たいので，多少の説明を加える。

VBを起動して付属CD-ROM（BLACK）のDllTest.vbpを読み込み，ソースコードを見ながら読んでいってほしい。

■Xファイル読み込み/表示

最低限の流れを書くと以下のようになる。

```
Call DX_Init(Pic.hWnd, 1)
MeID = AddMesh(MeshName, -1, 1)
Call ResetCamera(-1)
Call DX_Draw
```

この4つの命令を実行するだけで，VBのピクチャーボックスコントロールにXファイルが表示される。それぞれの意味を説明すると，

1行目：Direct3Dリテインドモード初期化

ピクチャーボックスのハンドルを第1引数，ハ

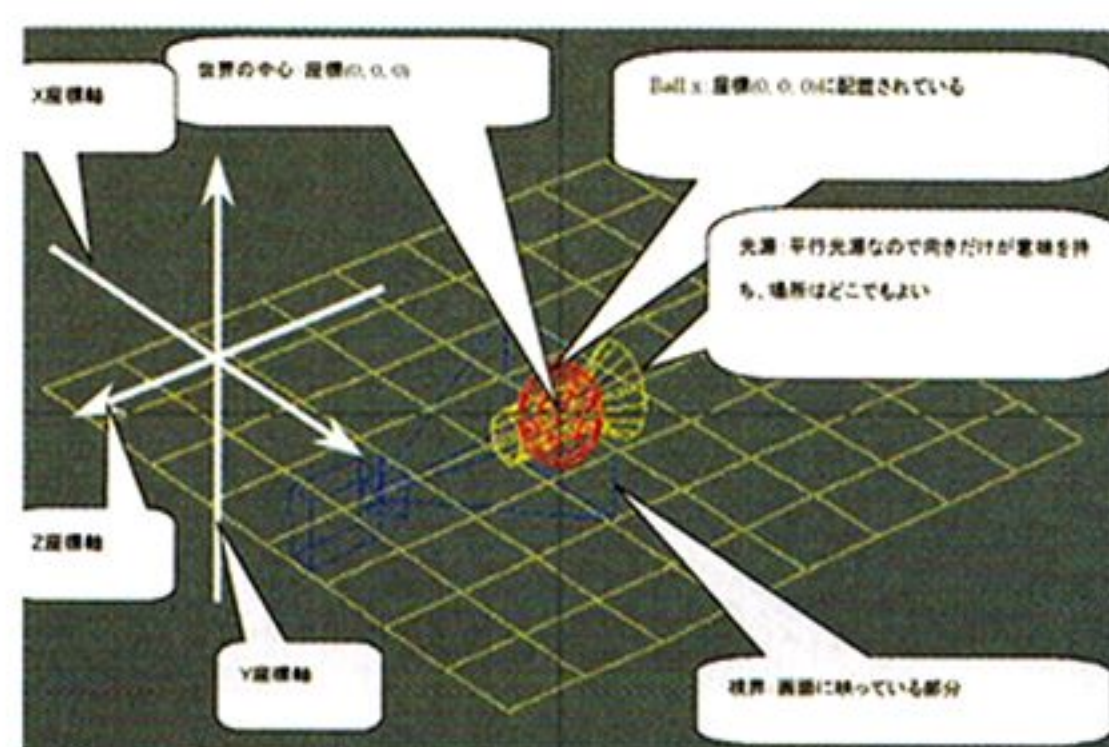



図1　カメラの座標系

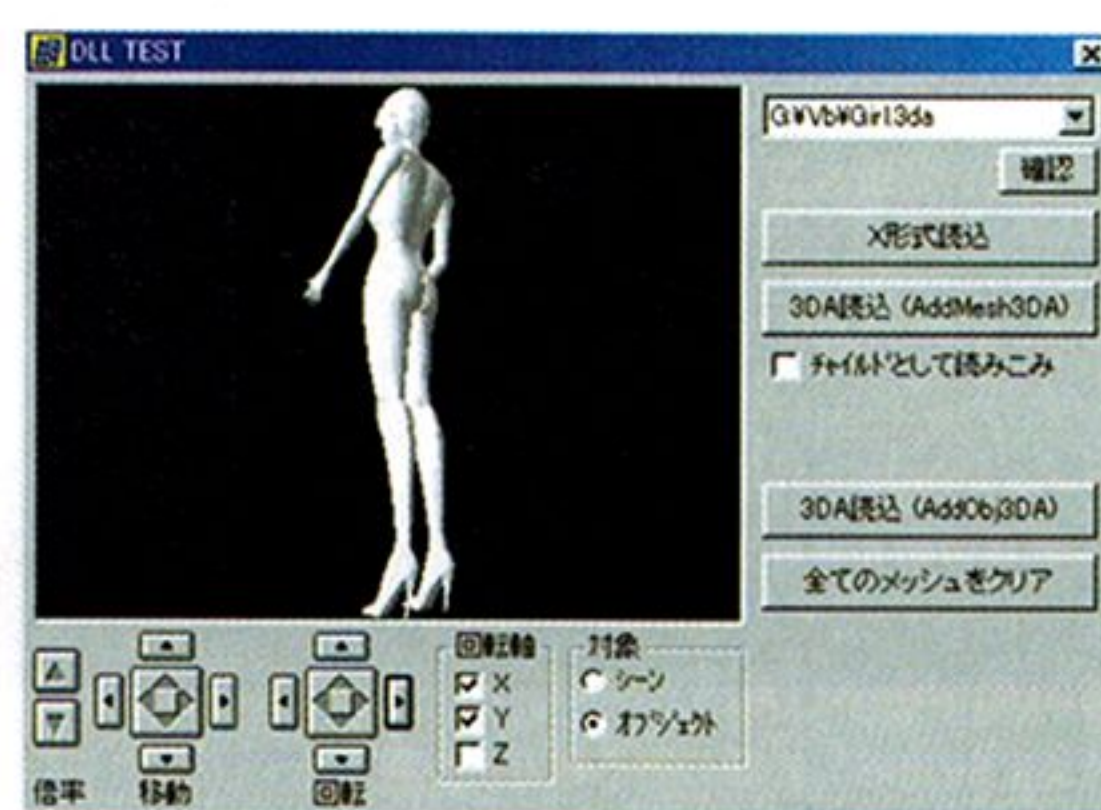



図2　ライブラリのテストプログラム

ードウェア使用スイッチONを第2引数に与えている。これは最初に一度だけコールすればいい。

2行目：メッシュをシーンに追加

Xファイル名を第1引数，親なしを第2引数，フラットシェーディング表示指定を第3引数に与えている。

3行目：カメラ位置の調整

対象オブジェクトに「シーン中のすべて」を指定して，カメラ位置を調整する。カメラ座標はXYは0でZは全体が収まるようなZのプラスの方向にセットされる。

4行目：レンダリング

2，3行目で設定された内容に従って，ピクチャーボックスにシーンを表示する。

Com_Xopen_Clickにソースが記述されているので参考にしてほしい。ここでは上記の2，3，4の処理のほかに，回転の中心座標の設定や移動コマンドで使用される移動量などの計算も行っている。

X形式で読み込むと，RendCtrlで設定した色，スペキュラー，半透明，テクスチャ情報などが反映されるので，綺麗な絵を表示できる半面，内部のオブジェクトが必ず1個として読みこまれるので，ツールなどを作りたいときには不便である。

3DAファイル読み込み（AddMesh3DA）

3DA形式は3Dアトリエのモデル用オリジナルフォーマットである。この場合色はファイル単位でしかつけられないので，ゲームなどを作るには不向きであるが，ファイル変換などを必要とせずに，モデルデータをそのまま表示できるため便利である。

読み込み〜表示の流れはXファイルと同様なので，Com_3DaOpen1_Clickと併用して流れを理解してほしい。

3DAオブジェクト読み込み（AddObj3DA）

この関数は内部に含まれているオブジェクト単位で3DAファイルを扱うことができるので，ツール制作にはもっとも適しているかもしれない。その気になれば簡易キャラモーションエディタ程度なら作れるはずである（というか元はといえば

VisualBasic用3D関数

COLUMN

DX_Init(hWnd, HardFlg)

HWnd As Long：

ピクチャーボックスのプロパティのhWndを与えればよい。

HardFlg As Long：

0：すべてソフトウエアで表示する

1：ハードウエアを使用する

基本的にこのパラメータは1を使う。表示がうまくいかない場合は，まずディスプレイドライバを最新のものに入れ替え，それでもだめな場合は，0を使う。

DX_Done()

DirectXが確保したすべてのオブジェクトおよび，DLL内で確保したすべてのメモリを解放する。

DX_Draw()

DLLの各関数で設定した内容に従ってシーンをレンダリングする。

ResetCamera(ID)

ID As Long：メッシュの管理番号

IDで指定されたメッシュが画面内に収まるようにカメラの位置を調整する。IDの値は，AddMesh，AddMesh3DA及びAddObj3DAの戻り値を指定する。IDに－1が設定されると，シーン中のすべてのメッシュが対象になる。

SetView(Projection, vField)

Projection As Long：

0：平行投影

1：パース投影

VField As Single：フィールドのサイズ

平行投影，パース投影の切り替えを行う。

SetCam_A(Cam)

Cam As PosAng：

ユーザー定義型の変数で，XYZ座標と回転角度を示す。

```
Type PosAng
  Px As Single    ：X座標
  Py As Single    ：Y座標
  Pz As Single    ：Z座標
  Ax As Single    ：X回転角
  Ay As Single    ：Y回転角
  Az As Single    ：Z回転角
End TypePosAng
```

Cam.Px = 0のようにして使用する。

カメラの位置と回転角度をワールド座標系の絶対値で設定する。

SetCam_R(Cam)

Cam As PosAng：

カメラの位置と回転角度を相対値で設定する。

GetCam(Cam)

Cam As PosAng：

カメラの現在の位置と回転角度のワールド座標系の絶対値で取得する。

SetLight_A(Lno, Lig)

Lno As Long：光源の番号

0：　正面からの光源

1：　背後からの光源

Lig As PosAng：

位置と角度を示すが，平行光源であるため位置情報には意味はない。同時に光源であるからZの回転角度にも意味はない。

光源の角度を設定する。

SetLightBrightness(Lno&, B)

Lno As Long：光源番号

B As Single：光源の明るさで1が最大

光源の明るさを設定する。

SetAmbient(A)

A As Single：光源の明るさで1が最大。

アンビエント（環境光）の強さを設定する。

ClearMesh(ID)

ID As Long：メッシュ管理番号

シーン中に読み込まれているメッシュを破棄する。IDに－1を指定するとどれかひとつが破棄される。

戻り値が－1の場合は，シーン中にメッシュが存在していないことを示す。

AddMesh(Fn, Parent, Q)

Fn As String：ファイル名

Parent As Long：

Q As Long：

レンダリングクオリティ

0：ワイヤーフレーム

1：フラットシェーディング

2：グローシェーディング

X形式のメッシュをシーンに読み込む。戻り値はメッシュ管理番号（ID）になる。色，スペキュラー，テクスチャなどが表示に反映される。

SetBGimage(Fn)

Fn As String：ファイル名

BMPかPPMフォーマットを指定すること。

背景イメージを読み込む。サイズは2のべき乗のサイズでなければならない。上下が逆に表示されるので，オリジナルの画像はあらかじめ上下反転しておく必要がある。2のべき乗とは，2，4，8，16，32，64，128，256…….である。

AddMesh3DA(Fn, Parent, Q)

Fn As String：

Parent As Long：

Q As Long：

3DA形式のメッシュをシーン中に1個のメッシュとして読み込む。戻り値はメッシュ管理番号。

AddObj3DA(Fno, ID, Parent, Q)

Fno As Long：ファイル番号

ID As Long：

そのためにこのDLLは作られたのである)。
　読み込みの流れは,
　1　ファイルオープン
　2　オブジェクト情報取得
　3　オブジェクトをシーンに追加
　4　ファイルクローズ
である。Com_3DaOpen2_Clickのコードを参考にしてほしい。

■注意事項

　移動/回転ボタンにはSheridan 3D Controlを使用している。これはVB4には標準でついていたコントロールである。しかし5になってからなくなってしまっているはずである。もしもあなたがVB5を初めてインストールし,3Dアトリエをインストールしていない場合は問題が起こる可能性がある。
　もうひとつ,Windows98がインストールされていると,Dx_Doneという関数を実行すると「不正な処理……」が出ることがある。実は今回の記事を掲載させていただくにあたり,プログラム部分の締め切りが早く訪れ,その時点ではその問題に気がつかず,いまこうして文章を書いている時点で判明して,あとの祭り状態だったりするのである(編注:編集部のマシンでは不具合は確認されていない)。
　いいわけはともかくとして,Win98に同梱されるDirectX5のバージョンが5.01になって,内部の仕様が変わってしまったらしい。だから5.00を使っている人は問題ないが,Win98および5.01を使っている人はhttp://micronetclub.co.jp/から対応版をダウンロードしてほしい。いや,この件に関しては私のミスであるので,このような威張った書き方はよろしくない。
　「どうか皆さん,面倒くさがらずにぜひとも対
　応版をダウンロードしてください」
なのである。

　3DAファイル内でのオブジェクト番号。
Parent As Long:
Q As Long:
　Open3DAを使って開かれている3DA形式のファイルから,指定したオブジェクトをシーン中に読み込む。戻り値はメッシュ管理番号。

Open3DA(Fn, Fno)
Fn As String:3DAファイル名
Fno As Long:ファイル番号。(0~99)
　AddObj3DAで使用する3DAファイルをオープンする。

Close3DA()
　Open3DAでオープンしたファイルを閉じる。

DX_SetColor(ID, R, G, B)
ID As Long:メッシュ管理番号
R As Single:
G As Single:
B As Single:
　赤緑青それぞれの成分で最大は1。
　シーン中に読み込まれているメッシュに色を設定する。

RendQuality(ID, Q)
ID As Long:メッシュ管理番号
Q As Long:レンダリングクオリティ
　すでにシーン中に読み込まれているメッシュのレンダリング方法を変更する。

AddBox(Parent, Siz)
Parent As Long:
　親メッシュ番号。-1を指定すると親なし。
Siz As MinMax:
　XYZの最大最小を表すユーザー定義型変数。
Type MinMax:
　minX As Single
　minY As Single
　minZ As Single
　maxX As Single
　maxY As Single
　maxZ As Single
End Type:
　シーン中にSizで指定されたボックスをワイヤーフレームで表示する。このボックスもひとつのメッシュであるため,戻り値はメッシュ管理番号を返す。

Move_A(ID, X, Y, z)
ID As Long:メッシュ管理番号
x As Single:
y As Single:
z As Single:
　ワールド座標系でのxyz絶対座標。
　IDで指定したメッシュを絶対座標で移動する。

Move_R(ID, X, Y, z)
ID As Long:メッシュ管理番号
x As Single:
y As Single:
z As Single:
　ワールド座標系でのxyz絶対座標。
　IDで指定したメッシュを相対座標で移動する。

Rotate_A(ID, X, Y, z)
ID As Long:メッシュ管理番号
x As Single:
y As Single:
z As Single:xyz回転角度
0~360などのように角度で指定する
　IDで指定したメッシュを絶対値で回転する。

Rotate_R(ID, X, Y, z)
ID As Long:メッシュ管理番号
x As Single:
y As Single:
z As Single:xyz回転角度
0~360などのように角度で指定する
　IDで指定したメッシュを相対値で回転する。

Scale_R(ID, X, Y, z)
ID As Long:メッシュ管理番号
x As Single:
y As Single:
z As Single:
　相対スケール値
　　IDで指定したメッシュを相対値で拡大/縮小する。

SetPivot(ID, X, Y, z)
ID As Long:メッシュ管理番号
x As Single:
y As Single:
z As Single:xyz絶対座標
　IDで指定したメッシュの回転の中心座標を設定する。

GetObjMinMax(ID, Obj)
ID As Long:メッシュ管理番号
Obj As MinMax:戻り値を受け取る変数
　XYZそれぞれの座標軸における最大値と最小値を得る。

GetObjPosAng(ID, Obj)
ID As Long:メッシュ管理番号
Obj As PosAng:戻り値を受け取る変数
　指定オブジェクトの現在の絶対座標および回転角度を得る。

Get3DAInfo(Fno, Oid, Oinfo)
Fno As Long
　Open3DAで指定したファイル番号。
Oid As Long
　3DAファイル内でのオブジェクト番号。
Oinfo As ObjectInfo
　上記2つのパラメータによって指定されたオブジェクトの情報。ユーザー定義型変数。
Type ObjectInfo
　MeID As Long
　Vst As Long
　Pst As Long
　Vno As Long
　Pno As Long
　Fn As String * 64
End Type
MeID:　メッシュ管理番号
Vst:　頂点開始番号
Pst:　面開始番号
Vno:　頂点数
Pno:　面数
Fn:　オブジェクト名
　3DAファイルを表示している場合は上記の情報を得ることができる。

Quake2でC言語のスキルアップ

須藤芳政/Sudou Yoshimasa

海外でいちばん人気のパソコンゲームといえば，なんといってもQuakeシリーズだろう。この手のダンジョン型対戦銃器撃ちまくりゲームは海の彼方の人々の心をとらえるらしく，いまだに類似ゲームがあとを絶たない。Quakeはオフィシャルの改造ツールやゲームソース公開など，別の意味である種の人々をとらえて離さない魅力を持っている。国内有数のQuake野郎に改造のイロハを語ってもらおう。

時代は常に流れゆくもので浮浪者ほどの社会的地位しか持っていなかった私もいまでは社会人1年生という恵まれた状態にあり，1着しかないスーツとYシャツでガンガン出勤しております。先日，絹100%のネクタイを洗濯機でグリグリに洗いまくり，乾燥機で20分ほど転がしておいたところキツネのしっぽのようにふっくらとした絶望的な仕上がりになってしまいました。注意せよ，社会人1年生！　ナイス忠告，私！

さて，id Software社のQuakeがリリースされたのは1996年ですから，もう2年が経つのですね，去年の暮れにはQuake2がリリースされました。この2年間私のPCゲーム総プレイ時間の99%はQuakeが占めており，通信対戦に首ったけ状態の私は電話を長時間ブッ続けでQuakeのために繋ぎっぱなしにしたため「アンタのところはいつ電話をかけても話中だ！」と姉が激怒，そういえば同じ多摩に住んでいるのにここ何年か会ったことがありませんね，お姉様，お元気ですか？

■私と改造モジュール

Quake，Quake2は通信対戦で遊ぶのが当たり前，ゲームは改造モジュール（オリジナルのQuakeに改造を加えたもの）が当たり前……という環境で過ごしてきた私は少々インターネットボケしているため，Quake2所有者の何パーセントがインターネットへ接続できる環境を持っているのかゼーンゼンわっかりません。ですが，私は日本のどこかに「コレがQuake2でゲスか？　ホホー，絵が綺麗ですネ，怪物イッパイ出てきて面白かったネ，デカキャラも倒したネ（デカキャラといえばZainソフトの「未来」ですな），ハイおしまいネ……」と物置に収納してしまう恐ろしい人々がいるのではないかとあまりの恐怖に声をあげずにいられないのです。「キャー！痴漢ー！」

Quakeが発売された当時，私はいまと同じくインターネット経由の通信対戦に夢中でウホウホいっておりました（プロバイダが遠くて電話代が3カ月で15万円オーバー，ひと月目で気づいて別のプロバイダ探せっちゅーの，私ってバカ）。そのうち誘導ミサイルや，壁に反射しながら飛ぶ炎，広範囲にわたって自分も含めたプレイヤーが大量爆死するという狂気じみた武器の使用が可能な海外Quakeサーバの出現に「どうなってんだ？Quakeの実行ファイルを逆アセンブラとかバイナリエディタとかイ・ケ・ナ・イ手段でコード編集してんのか？　ヒョー！」と私のインチキ改造魂に火が入りました。

いろいろ情報収集を行った結果，Quakeには QuakeCコンパイラなるものがあらかじめ用意されており，ユーザーがQuakeCというCまがいの言語を使用して自由にゲーム内容を改造できる仕様になっていたのです。「作った会社が改造を奨励してんのか，このゲーム正解やで（パソピア7語録より一部参照），人気絶頂や！（パソピア7語録より抜粋）」と狂喜して飛びついたものです。

■Quake2のソース，クレヨ！

Quake2発売を目前にした去年の暮れ，私にとって敵キャラクター，サウンド，グラフィック，マップなどはもはやどうでもよく，Quake2Cの仕様公開のみが狙いで待ち受けていたのであります。そして，Quake2のソースを手に入れた私は驚きのあまりプロテイン入りの牛乳をドチャーっと畳にぶちまけてしまいました。ゴキブリ事件も含め，すでに3枚も畳を損壊している私，ノープロブレム！　ドンマイ大家さん！　ポジティブで行こう！（行けねーよ）

……で，Quake2のソースセットはCのソースファイル（C++ではありませんCです）とともにVC++5.0のプロジェクトファイルが付属，ビルド一発でオリジナルとコンパチのゲームモジュールができあがりっと……あとは自分の好きなようにソースに改造を加えてゆけば改造モジュールのできあがりっと……こんな感じでQuakeCのようなインチキC言語ではなく，本物のCからビルドしたDLLを使ってQuake2のゲームを動かせるようになっていたのです。

で，そのモジュールってどのファイルなのか教えちくりよ！　という話になりますね。Quake2インストール先ディレクトリ中のbaseq2ディレクトリにあるgamex86.dllというファイルがソレであります。

ひょっとしたらこんな人がいるかも？「VC++5.0なんて持ってネーヨ！　ビルドデキネーヨ！Quake2，持ってネエヨ！」という人にアドヴァイス，「買え」。

まあ，気合があればVC++4.0でしょうがBorand（名前変わった？）でしょうが，Watcomでしょうが，SYMANTECでしょうがフリーウェアのCコンパイラ（少々やっかいな後処理が必要なものあり）でしょうが，GetGameAPIをエクスポート関数に持つDLLがビルドできれば問題なーし……かもしれないのでチャレンジしてみるのもいいかもしれませんね（VC++4.0でのビルドはできましたヨ，というよりできないとヤバいです）。

■絶対必要なもの

Quake2改造モジュールのプログラミングを行うにあたり，すでにC言語を完全にマスターしてるヨって人は問題なし，逆に私のお話なんぞ聞いてたら片腹痛くてヘソが茶を沸かすわタワケがなんて思うかも。ある程度使えるヨって人はスキルアップが望めるかもしれません。C言語の勉強は始めたばっかりだヨって人にもオススメしたいですな〜。

「Hello World」から始まって2，3個サンプルプログラムを実行したあとの言語演習なんてはっきりいってサンプルリストの打ち込みが面倒なだけでクソだとは思いませんか？（超問題発言!?）

私は自分の興味のある既存のプログラムソースをC言語の手引きやビルダのヘルプファイルを駆使して解析しながらいじくり回したほうが楽しく学べると思うのですけれど。

で，今回のプログラミングに必要なものはほとんどインターネット上から入手しなければなりません。CD-ROMに全部入れてしまうのがインターネット接続環境のない人にとって都合がよいのでしょうけれど「チミは，CD-ROMなどのメディアへ，このソースコードをコピーして配布してはイケナイ」といった内容のテキストがソースファイルのアーカイブに付属していたので怖くて配布できません。「そんなら，いったいどーしたらよいのでしょーか？」と絶望感に見舞われた人はインターネット接続環境を持つ友人に「頼むっからダウンロードしてク・レ・ヨ！」と半ば脅し入りで頼み込むか，会社で専用線を引いているような恵

まれた環境の場合は昼休みにダウンロード後，フロッピーへ分割して持ち帰るといった手段でなんとか入手してください。

まずはQuake2のv3.17パッチが必要です(厳密にはv3.14以上であれば可)。このパッチはid SoftwareがQuake2発売以降バージョンアップを行うたびに配布してきたもので，最新が7月現在でv3.17なのです。パッチの存在を知っている人はもうすでに各自v3.14以上までバージョンアップさせていることでしょう。それ以外の人はおそらく発売直後のバージョンv3.05のままだと思います。Quake2起動後〝半角/全角〟キーを押して表示されるコマンド入力待ち画面(コンソール画面)の右下にバージョン番号が記されているので確認を。

次に肝心のソース。このソースをコンパイルして得られるDLLはv3.14以降のQuake2とともに動作させることができます。v3.14よりも古いバージョンで動かすことはできません。v3.14よりも古いバージョンのQuake2で動かすことのできるDLLのソースもまだidのftpサイトに置いてあるようですが，どうせなら新しいほうがいいでしょう(この業界，ヴィンテージ志向なんてタダの化石だ！　あ，でも使ってるわX68000)。

新旧の違いは，構造体のサイズです。関数のアドレスを格納している構造体も存在するため，たいていの場合無理やり動かすと「このプログラムは不正な処理を実行しました」でお亡くなりになってしまうので，それを回避するためにAPIバージョンの設定がソースのヘッダで定義されています。Quake2起動時に「Error during initialization」と表示されたらバージョンが違うということですな。以下に上記2ファイルのアドレスを記すのでなんとかゲットしていただきたい。いや，ゲットしろ！　私の書く意味がなくなっちゃう。

●V3.17へフルバージョンアップするためのパッチ

ftp://ftp.idsoftware.com/idstuff/quake2/q2-317-x86-full.exe

●バージョン3.14のソース

ftp://ftp.idsoftware.com/idstuff/quake2/source/q2-3.14-source.exe

■Quake2モジュールの動作

実際にビルドを行う前に，ここでQuake2本体プログラムとgamex86.dllがどのようなリンクを行って動作しているのか簡単に説明しちゃいましょう。

まずQuake2が起動された際，Quake2からgamex86.dllが動的に読み込まれ，その後GetGameAPI関数が呼ばれます。

Quake2はGetGameAPI関数に対し，空間のトレース，サウンドの再生，メッセージの表示など，DLL側で必要な関数のアドレスを格納しているgame_import_t構造体のポインタを引数として渡し，その構造体の内容はDLL側でgame_import_t構造体giへ取り込まれます。

その結果，Quake2側の関数を〝gi.〜〟という形で実行できるようになるのです。深く考える必要なし，「gi.を頭につけりゃ便利な関数使い放題ヤッホー」(使い方は自分で見つけなきゃならないけどね)それだけでいいじゃないですか。これらをインポート関数と呼びましょう。

DLLはQuake2に対し，APIバージョン，DLL内でQuake2から直接呼び出されるDLL内関数の各アドレスなどをgame_export_t構造体へ格納し，構造体のポインタを戻り値として返しています。これらの関数はエクスポート関数と呼びましょう。

上記インポート，エクスポート関数によってQuake2とDLL間の入出力は実現されているんですね。GetGameAPI関数の処理はソースファイルg_main.cに記されています。

Quake2はエクスポート関数を必要に応じて呼び出します。DLLはそれに応じた処理を規定の関数内に記さなければなりません。これらの関数を簡単に紹介しましょう。

DLL読み込み直後にはInitGameが呼び出され，メモリの確保，コンソール値の初期値設定(Quake2はあるキーワードに対して数値を設定することにより，ゲーム中の処理にその値を反映させています。たとえばゲーム難易度skillの初期値は1)が行われています。逆にゲームのシャットダウン(メモリの解放処理)時はShutdownGameが呼び出されます。

ゲーム中にマップが変更になった場合には情報退避，再度取得のためにWriteGame，ReadGame，WriteLevel，ReadLevelが使用されているようです。

ゲームが新たなマップへ移動した場合，ドア，プラット，ボタン，アイテム，モンスターその他を配置および初期化処理を施すために呼び出されるのはSpawnEntities。

クライアント接続時の処理はClientConect，ゲームへの参加はClientBegin，クライアントの切断はClientDisconnect(Quake2は，たとえスタンドアロン状態のマシンでプレイヤーが1名だったとしても，それはサーバへ1名のプレイヤーが接続しているのだという感覚で扱ったほうがよいでしょう)。クライアントの名前やモデル，スキンの変更時はClientUserinfoChangedが呼び出されます。

味方と一緒に2対1で撃ちまくります。正々堂々の勝負が爽やかです

ClientThinkは主にクライアントの入力(マウス，ジョイスティック，キーボード)による移動情報をゲームに反映させ，ClientCommandはクライアントコマンドの処理(チャットコマンドその他)を行います。毎フレームごとに呼出されます。

ServerCommandはサーバ専用のコマンド処理のために用意されたClientCommandと扱い方は似ていますが，クライアントからでは実行できないという点が異なります。

G_RunFrameは毎フレームごとに呼び出され，マップ内に存在するドア，プラット，ボタン，アイテム，モンスターその他の各処理を呼び出しています。

「〜ます」ばっかりでマス大山を思い出してしまいました。私も1撃で牛を倒せるようになりたいですな。ンモー。

■ビルドして実行してみたい

ビルドを行うのは簡単です。VCのビルドコマンドを「ポチっとな」ってな感じで実行すればジャジャーンとgamex86.dllができあがってしまうのですから。問題はQuake2実行時にビルドしたDLLをどう読み込ませるかですな。「俺ぁ江戸っ子だからヨ，できあがりやがったDLLをそのまんまbaseq2ディレクトリにチョチョイってな具合に上書きOKで放りこんだら即，一発実行よ，おう，イキだろ？」……絶対にやってはイ・ケ・マ・セ・ン！　自分でビルドしたDLLはQuake2インストール先のbaseq2ディレクトリと同レベルのパス下へ新たにディレクトリを作成して，そこへ置きます。

たとえばQuake2のインストール先がC:¥Quake2ならば以下のように好きな名前で新規ディレクトリを作成しましょう(〝nobita〟はあくまでも例です)。

C:¥Quake2¥nobita

しかし，このディレクトリにgamex86.dllをコピーしただけでいままでと同様にQuake2を起動していたのでは，当然いままでと同じようにQuake2はbaseq2ディレクトリ内のgamex86.dllを読み込んで実行してしまいます。許せませんね。そこで，Quake2.exeのショートカットを作成し，setスイッチでgameキーワードへディレクトリ名の設定を行います。これでQuake2はnobitaディレクトリ下のgamex86.dllを読み込んでくれるのですよ。

C:¥Quake2¥quake2.exe +set game nobita

ついでにやっておきたいことは，毎回ビルドするたびにDLLをnobitaディレクトリにコピーしていたのでは面倒臭くてやってらんないので，VCのプロジェクトの設定からリンクタブを選択し，出力ファイルを以下のように直接指定しちゃいましょう。

C:¥Quake2¥nobita¥gamex86.dll

わーい！　これでチミもイッパシのQuake2デベロ野郎or嬢だ！　自信を持って実家へお便りしてもイイゾ！　「探さないでください」

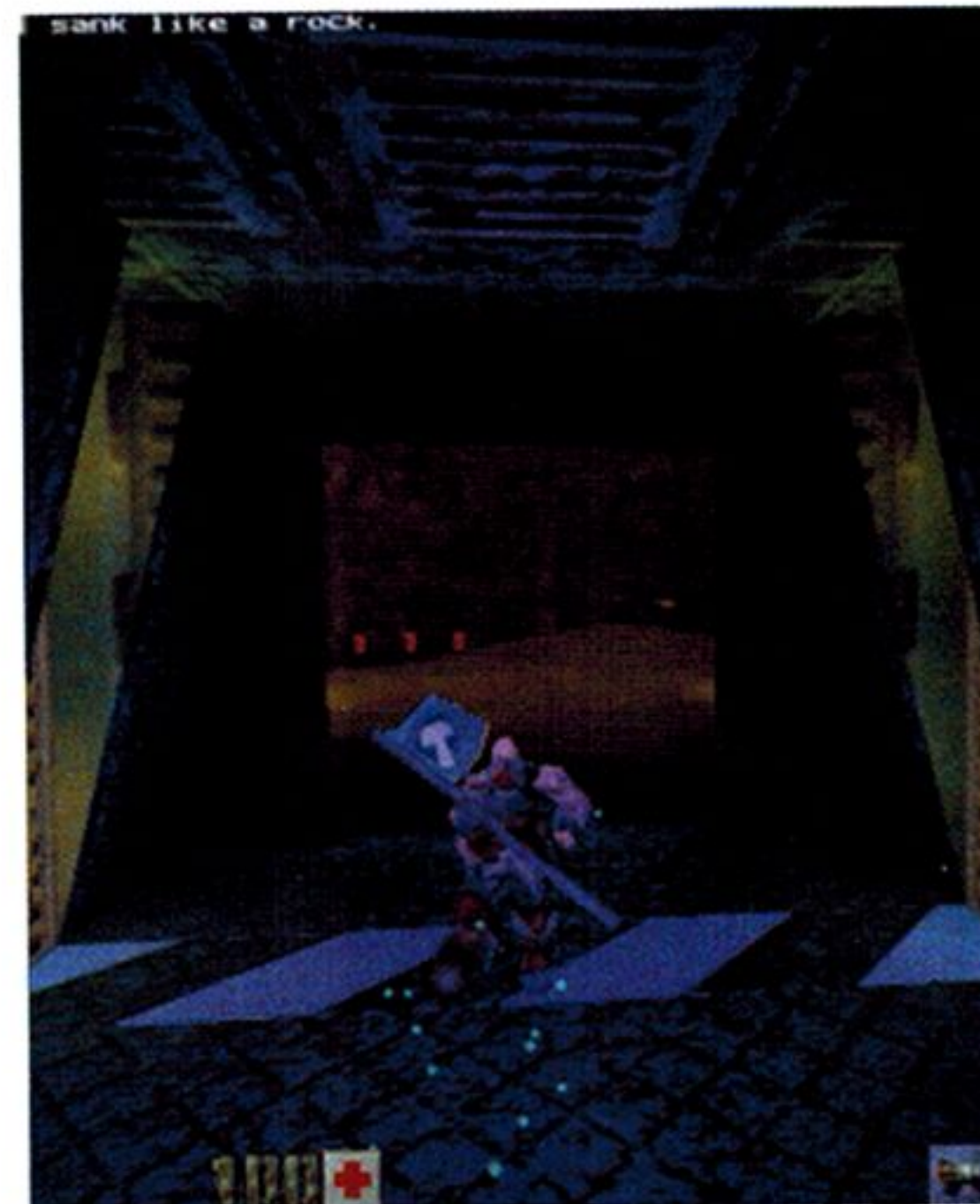
■最後に

今回はこれでおしまいです。また機会はあるのかな？　ほかにも画面のレイアウト変更とかコンソール値の新規追加とか，いーっぱい紹介したい事柄があるのですけど，自ら解析して発見できなくはないものなので，ひと頑張りしてみるのもスキルアップに繋がるのでいいことかもしれませんね。

改造モジュールの例としてはCTF（各プレイヤーがチームに分かれて旗取り合戦），Bot（プレイヤーの代わりにロボットが対戦の相手をしてくれる），その他，挙げればキリがありません。異端なモノでは戦闘機で撃ち合うものまであり，3Dゲームならなんでもイケんじゃないか？　とさえ思えます。ちなみに私はBotを作ってみました。CD-ROMに入れてあります。

前に外人さんから「Quake2ソースのC言語は英語で書かれているのになぜ英語が片言な日本人の貴方はプログラムが組めるのか？」と質問されたことがある。どうやら日本だとC言語は「モシ ～ ナラバ ～ ヘイケ」のような記述をしていると勘違いしていたらしいのです。世界は広いですな。私は「ぴゅう太」を思い出しちゃいましたよ。

私はこの前Z氏のお宅へおじゃましたときに「キミはQuakeの対戦で入るとき，なんて名前で入っているの？」と聞かれて「えー，“ponpoko”で入っています」と，とっさに答えてしまいましたが本当は“(_̀)/Axxx-Cxxx”という完全なるタブーワードで世界に君臨しているのでした。すいません。

自分だけ安全なところから狙い撃つ。男の子の基本ですね

この人，旗守ってますね。無駄な抵抗いけませんね

Quakeひとくちメモ

COLUMN

Quakeについてなにも知らないキミのために親切心丸出しな露出王の私は説明せずにはいられないのでした。

1996年にid Software社が1作目Quakeをリリースするまでは同社のDOOMシリーズ，3D Realms社のDUKE NUKEM 3Dがこのテのシューティングでは猛威を振るっていたようですが，敵キャラクターは2D表示で薄っぺらいダンボールのハリボテがガウガウ吠えながら攻撃してくるという，PSやSSでポリゴン中毒気味な現代のお子様がプレイしようものなら「PCなんてダッセーよな」などと発言しかねないものでした。

しかし，Quakeは一部の爆発表示など以外はすべてポリゴンを使用し，キャラクターのグラフィックデザイン，悲鳴などの効果音は私的にいうとかなり馬鹿度が高めであるうえ，最大16人がTCP/IPのネットワーク環境で対戦可能（現在は32人近くで対戦する方法もある），QuakeCというCライクな言語でゲームのプログラミングが可能，挙げ句にはマップエディタやモデラをユーザーがガンガン作ってしまう始末。さらにアメリカのQuake大会優勝者にフェラーリ進呈という驚きのゲームでした。

続いて1997年末にリリースされたのがQuake2というワケです。キャラクターのグラフィックや効果音がカッコよくなり，反して馬鹿度が極度に落ち込んでしまったのがちょっと残念。前作の「末はガッツか勝新か」といわんばかりのヘヴィーな圧迫感を与える主人公のグラフィックデザインが女性に大不評だったのではないか？　と疑ってみたくなります。Quake2になってから女性キャラも選択できるようになった点が実に怪しいです。

Quakeシリーズはシングルプレイヤーモードよりも対戦サーバ経由のマルチプレイヤーモードが主流だと私は超勝手に考えております。世界は今回触れた改造モジュールのサーバで溢れており，「フツーじゃ満足できねえYO！」という輩がロケットランチャーをブッ放しまくり。チャットで汚い言葉を飛ばしまくりの奴もいれば，妙にフレンドリーな奴もいます。チーム戦でじっとしたまま怠けている奴がいたのでロケットランチャーの爆風で派手にマグマに落っことしてやったら「なにすんだ！俺は一服してんだ！」と怒られちゃいました，テヘ。

というわけで，Quakeを持っていてインターネット環境があるならば，いますぐに出陣してク・レ・ヨ！

イマドキのPCゲーマーは3Dアクセラレータボードが必須。Quake，Quake2も3Dアクセラレータボードあればとっても幸せな気分になれます。私は今日に至るまでほとんどQuakeのためだけに自分のPCをひたすらアップグレードしてまいりました。当然Voodooボード搭載。もう低解像度なんかでプレイできない人間になってしまったのです。低解像度で思い出しましたけど，ドット絵職人という職種名は死語ですか？　「俺はデザイナーでも絵描きでもねぇ！ドット絵職人だ！」という生粋のドット絵職人さんがいたら伝書バトでお知らせください。

どうですか，Quakeシリーズは？　「マジQuakeなんて知らなくても～，別に～，マジ死ぬわけじゃないんで～，マジいらねっすよ」そのとおり。

ですが，私はハードに，そしてヘヴィーに推奨する！　「コンニャロ！Quake，シ・ロ・ヨ！」

先日仲間内で「小森のおばちゃまが夢中になっている3Dゲームとはなにか」という話題になったのですが，結局，語呂の関係により残念ながらQuakeではなくDUKE NUKEM 3Dという結論に達しました。「お，おばちゃまがイマいちばんハマっているのはねぇ，DUKE NUKEM」……ああ，なんてくっだらない話をしているんだお前らは。

Quake，Quake2情報は以下のサイトへ行けばいろいろ入手できるので，初心者の人はここからいろんなリンク先に飛んでみてね。

http://www.jp.tri6.net

以上，Quake2ひとくちメモでした（たぶん）。

Quake2インポート関数簡易リファレンス

本文でも触れたインポート関数はコードがDLLのソースへ書かれているわけではないので，各関数の使い道は自分で見つけるしかありません。でも，一部の関数については私がお助けしてさしあげま〜す！　本文で述べたとおり，各関数のアドレスはgame_import_t構造体giに格納されるので，インポート関数呼び出し時は関数の頭にgiをつけてください。以下にbprintf関数の使用例を示します。

```
  gi.bprintf(PRINT_HIGH,"Hellow World");

 ほかのインポート関数については自分で調べましょう(根気だ！)。

void            bprintf(int printlevel, char * fmt, ...);
 接続中のプレイヤー全員の画面に対してメッセージを書式付きで出力。
引数
printlevel      以下の中から選択
                    PRINT_LOW        アイテムを拾ったときのメッセージ用?(未使用)
                    PRINT_MEDIUM     死亡通知用
                    PRINT_HIGH       優先順位高
                    PRINT_CHAT       チャットメッセージ(BEEP音とともに緑色でメッセージを出力)
fmt                 書式制御文字列

void            dprintf(char * fmt, ...)
 デバッグ用メッセージを書式付きで出力。
引数
fmt             書式制御文字列

void            cprintf(edict_t * ent, int printlevel, char * fmt, ...)
 特定のプレイヤーの画面に対してメッセージを書式付きで出力。
 (プレイヤー以外に対して使用してはいけない)
引数
ent             メッセージを表示するプレイヤー
printlevel      bprintf関数に同じ
fmt              書式制御文字列

void            centerprintf(edict_t * ent, char * fmt, ...)
 特定のプレイヤーに対し，画面の中央へメッセージを書式付きで出力。
引数
ent             メッセージを表示するプレイヤー
fmt             書式制御文字列

void            sound(edict_t * ent, int channel, int soundindex,float volume, float attenuation, float timeofs)
 指定したプレイヤーに効果音を発生させる。
引数
ent             効果音を発するプレイヤー
channel         効果音の再生チャンネル，以下の中から選択
                    CHAN_AUTO        自動選択
                    CHAN_WEAPON      武器用
                    CHAN_VOICE       声用
                    CHAN_ITEM        アイテム用
                    CHAN_BODY        体用(水に飛び込んだときなどの効果音)
Soundindex      効果音のindex番号を指定する。index番号は同じインポート関数であるsoundindex関数によって得る
volume                               0.0〜1.0の間で再生ボリュームを指定
attenuation     効果音の減衰タイプを以下の中から選択
                    ATTN_NONE        減衰無しでそのまま再生
                    ATTN_NORM        普通
                    ATTN_IDLE        ?
                    ATTN_STATIC      遠距離で聞いた場合，減衰が著しくなる設定
timeofs             WAVファイルの再生開始ポイント，通常0を指定

void            positioned_sound(vec3_t origin, edict_t * ent, int channel,
                                 int soundinedex, float volume, float attenuation, float timeofs)
 指定した地点で効果音を発せさせる。
引数
origin          効果音の発生地点
ent             効果音を発するentity
channel         sound関数に同じ
soundindex      sound関数に同じ
volume          sound関数に同じ
attenuation     sound関数に同じ
timeofs         sound関数に同じ

void            configstring(int num, char * string)
 各種設定番号およびindex番号へ文字列データを割り当てる。
引数
num             対象となる各種設定番号およびindex番号
                CS_NAME              ゲームの名前?(0)
                CS_CDTRACK           再生するCDトラック(1)
                CS_SKY               空のタイプ名?(2)
                CS_SKYAXIS           空の回転軸位置?"%f %f %f"の形で指定(3)
                CS_SKYROTATE         空の回転タイプ?(4)
                CS_STATUSBAR         ステータスバーのレイアウト設定スクリプト(5)
                CS_MAPCHECKSUM       チートマップ摘発用のチェックサム?(31)

                CS_MODELS            model indexの先頭番号(32)
                CS_SOUNDS            sound indexの先頭番号(CS_MODELS+MAX_MODELS)
                CS_IMAGES            image indexの先頭番号(CS_SOUNDS+MAX_SOUNDS)
                CS_LIGHTS            light indexの先頭番号(CS_IMAGES+MAX_IMAGES)
                CS_ITEMS             item indexの先頭番号(CS_LIGHTS+MAX_LIGHTSTYLES)
                CS_PLAYERSKINS       player skin indexの先頭番号(CS_ITEMS+MAX_ITEMS)
                MAX_CONFIGSTRINGS     (CS_PLAYERSKINS+MAX_CLIENTS)

MAX_MODELS, MAX_SOUNDS, MAX_IMAGES, MAX_LIGHTSTYLES, MAX_ITEMS, MAX_CLIENTS
はいずれも現在の設定では256である

string          割り当てる文字列

＊model, sound, imageはそれぞれmodelindex, soundindex, imageindex関数
を使用して各index番号へファイル名を割り当てているので，直接configstringを
使用することはない。

void            error(char * fmt, ...)
 エラーメッセージを書式付きで出力。
引数
fmt             書式制御文字列

int             modelindex(char * name)
 指定されたmodel(md2ファイル)をキャッシュへ読み込むとともにindex番号を取得。キャッシュへすでに読み込まれている場合はindex番号を返す
のみである。
引数
name            modelファイル名(相対パス入りで指定)

戻り値
                modelのindex番号

int             soundindex(char * name)
 指定されたsound(wavファイル)をキャッシュへ読み込むとともにindex番号を取得。キャッシュへすでに読み込まれている場合はindex番号を返すの
みである。
引数
name            soundファイル名(相対パス入りで指定)
戻り値
                soundのindex番号

int             imageindex(char * name)
 指定されたimage(pcxファイル)をキャッシュへ読み込むとともにindex番号を取得。キャッシュへすでに読み込まれている場合はindex番号を返すの
みである。
引数
name            imageファイル名(相対パス入りで指定)
戻り値
                imageのindex番号

void            setmodel(edict_t * ent, char * name)
 特定のentityへmodelを割り当てる。
引数
ent             modelを割り当てるentity(プレイヤー，モンスター，武器，その他)
name            modelファイル名(相対パス入りで指定)

trace_t         trace(vec3_t start, vec3_t mins, vec3_t maxs, vec3_t end,edict_t * passent, int contentmask)
 指定サイズのBOXで特定区間のトレーシングを行う。トレーシング中に対象となる
コンテンツへBOXが衝突するか，トレース区間の終端へ到達すると処理は終了する。
引数
start           トレース開始地点
mins            トレースBOXの最下位相対座標
maxs            トレースBOXの最上位相対座標
end             トレース終了地点
passent         トレース中無視するentity(普通は自分自身を指定する)
contentmask     以下のコンテンツからトレース対象となるものを選択し，ビットをONにする。
                CONTENTS_SOLID              固体(1)
                CONTENTS_WINDOW             窓(2)
                CONTENTS_AUX                ?(4)
                CONTENTS_LAVA               溶岩(8)
                CONTENTS_SLIME              スライム(16)
                CONTENTS_WATER              水(32)
                CONTENTS_MIST               霧(64)
                LAST_VISIBLE_CONTENTS     ?(64)
 これより上位ビットにもコンテンツの種類が割り当てられていますが，q_shared.h
を参照してください。
戻り値
 以下のtrace_t構造体へトレース結果を得る

typedef struct{
                qboolean          allsolid;       // trueの場合，平面は無効(使われていない?)
                qboolean          startsolid;     // trueの場合，startで指定した場所は壁や岩など
                                                  // の塊の中だった
                float             fraction;       // 1.0だった場合トレース終了まで接触したものはない
                vec3_t            endpos;         // トレースを終了した地点
                cplane_t          plane;          // 接触面の情報
                csurface_t       * surface;       // 接触面表面に関する情報
                int                 contents;     // トレース終了地点のコンテンツタイプ
                struct edict_s      *ent;         // 接触したentity
} trace_t;

int             pointcontents(vec3_t point)
 指定した座標のコンテンツタイプを得る。
引数
point           コンテンツタイプを知りたい座標
戻り値
 以下に示すいずれかのコンテンツタイプを戻り値として得る
                CONTENTS_SOLID              固体(1)
                CONTENTS_WINDOW             窓(2)
                CONTENTS_AUX                ?(4)
                CONTENTS_LAVA               溶岩(8)
                CONTENTS_SLIME              スライム(16)
                CONTENTS_WATER              水(32)
                CONTENTS_MIST               霧(64)
                LAST_VISIBLE_CONTENTS     ?(64)

void            linkentity(edict_t * ent)
 entで指定したentityとほかのentityとの相互作用を更新する。
引数
ent             相互作用を更新したいentity

void            unlinkentity(edict_t * ent)
 entで指定したentityとほかのentityとの相互作用を解除する。
引数
ent             相互作用を解除したいentity

int             BoxEdicts(vec3_t mins, vec3_t maxs, edict_t **list,int maxcount, int areatype)
 特定box内に存在するentity数，およびリストを得る。
引数
mins            boxの最下位絶対座標
maxs            boxの最上位絶対座標
list            entityリストを受け取る配列
maxcount        受け取るentityリストの最大数普通MAX_EDICTSを指定
areatype        以下の2つから選択
                AREA_SOLID              playerやmonster，その他固体の物体
                AREA_TRIGGERS           アイテム，武器などを対象に行う
戻り値
 box内に存在するentity数

void            TagMalloc(int size, int tag)
 メモリ領域を確保。
引数
size            サイズの指定
tag             領域確保するタグを以下から選択
                TAG_GAME                  dllが読み込まれていなければ空(765)
                TAG_LEVEL                 新しいlevelを読み込む際は空(766)

void            FreeTags(int tag)
 指定されたタグのメモリ領域を解放。
引数
tag             領域開放するタグを以下から選択
                TAG_GAME                dllが読み込まれていなければ空(765)
                TAG_LEVEL               新しいlevelを読み込む際は空(766)

cvar_t          cvar(char * var_name, char * value, int flags)
 コンソール値を新規作成。たとえば，ゲームの難易度が設定されているコンソール値"skill"はデフォルト値"1"で作成されている。
引数
var_name        コンソール値の名前
value           コンソール値のデフォルト値であると思われる
flags           コンソール値の属性を指定(0を指定すると普通のコンソール値?)
                CVAR_ARCHIVE            セーブ時に設定される
                CVAR_USERINFO           変更時，ユーザー情報へ追加
                CVAR_SERVERINFO         変更時，サーバー情報へ追加
                CVAR_NOSET              コマンドラインからのみ変更可能でコンソールからは変更不可
                CVAR_LATCH              サーバーが再スタートするまで変更は温存
戻り値
 以下に示すcvar_t構造体へ設定した内容を得る

typedef struct{
                char              * name;             //コンソール値の名前
                char              * string;           //設定されている値(文字列)
                char              * latched_string;   //CVAR_LATCHを設定した場合用
                int               flags;              //コンソール値の属性
                qboolean          modified;           // コンソール値が変更されるたびにフラグはセットされる
                float               value;            //設定されている値 (数値)
                struct              cvar_s * next;    //おそらくコンソール値リスト上の次のコンソール値情報
} cvar_t;

cvar_t          cvar_set(char * var_name, char * value)
 コンソール値を設定。
引数
var_name        コンソール値の名前
value           設定する値(文字列)
戻り値
 cvarを参照

cvar_t          cvar_forceset(char * var_name, char * value)
 コンソール値を属性無視で強制的に設定(危険)。
引数
var_name        コンソール値の名前
value           設定する値(文字列)
戻り値
 cvarを参照

int             argc(void)
 クライアントコマンド"cmd"へ与えられたパラメータの数を得る。C言語のmain関数の引数と似ている。
戻り値
 クライアントコマンド"cmd"へ与えられたパラメータの数

char *          argv(int n)
 クライアントコマンド"cmd"へ与えられたパラメータを得る。C言語のmain関数の引数と似ている。
引数
n:              "cmd"へ与えられた何番目のパラメータの内容を得るか (0〜)
戻り値
 クライアントコマンド"cmd"へ与えられたn番目のパラメータ内容

char *          args(void)
 クライアントコマンド"cmd"へ与えられた1番目のパラメータよりも後ろの内容を得る。
たとえば"cmd gian suneo"であれば戻り値は"gian suneo"となる。
戻り値
 クライアントコマンド"cmd"へ与えられた1番目のパラメータよりも後ろの内容

void            AddCommandString(char * text)
 コマンド実行。
引数
text            コンソールのコマンドライン
```

ゲームデザインのススメ

清瀬栄介

ゲームを作るときにプログラムの技術論は多数出てきても，デザインの方法論についてはほとんど語られることがない。ここでは本質的なゲーム性のあり方について考えてみる。それを実現するための具体的な方法論も紹介する。

デザインがなぜ必要か？

たとえば，なにかゲームのアイデアを考えたとする。そのイメージがどんどん膨らんでいくときはすごくワクワクするに違いない。いままでのどんな作品よりスゴい。こりゃカッコいいぜ。

でもこれが，イザ形にしようと思うと，どうしていいかわからない。頑張ってペンをとってみても，書いたモノを読むとイメージと違う。おかしいなぁ，自分が考えているのはこんな陳腐なモノじゃないんだけど。そんなコトを思った経験はないだろうか。

あるいはプログラムができる人なら，ある程度イメージを形にしたかもしれない。思ったとおりのモノが動いてるんだが，なんだかイマイチしまらない。面白くない。どうすりゃいいんだろ？

そう思っているうちに，いつの間にかそのファイルをいじることもなくなってしまう。そんな体験は？

モノを作るのに情熱が大事なのはいうまでもない。でも，それだけでは超えられない壁があるのも確かだ。アナタがもし「妄想プランナー」，「やりかけプログラマ」で終わっているとしたら，アナタに足りないものはきっとゲームの性質の理解とゲームデザインの手法だ。

みんな，ゲームデザインについては語りたがらない。クリエイターのインタビューを読んでも，どんな風にアイデアをゲームにしたのかは，詳しく載っていることはない。

「結局手法なんて人それぞれで，できるヤツはできるんだよ」

というコトらしい。

まぁ確かに才能でやっちゃう人もいるし，自分の正解が他人に当てはまるかどうかはわからない。でも映画の脚本だって，書き方の本はいっぱい出ている。脚本の内容には才能が出るだろうが，才能がなきゃ書き上げることもできないってことはない。ゲームデザインだって同じはずだ。

僕は別に大ヒットゲームを作ったわけではないが，それでもゲーム業界に身を置く者として，いくつかコンテンツの企画もしたし，他人の企画案も見てきた。そこで感じていることをここにまとめて，少しでもゲームデザインの手法の確立に貢献できればと思う。

ゲームにする必要，あんの？

初めにいっておきたいのが，

「ゲームというメディアの本質を知らなければ，ゲームを作ることはできない」

ということだ。

1996年末に「パラッパラッパー」，「I.Q.」といった，ゲーム業界出身でない人が企画したゲームが大ヒットした。それ以降，ずいぶんとゲーム業界以外のクリエイターから「ゲームを作りたい」という相談を受けることが多くなった。

彼らの企画は，「こんなことができたら楽しい」，「こんな設定こんなデザインを考えた」というのが大半で，なかには「こんな人にウケたい」ということしか考えていないものもあった。

PlayStationの普及とともにゲームの地位が上がったのは確かだ。特に，上記の2作が出てからは，ゲームが一部の「濃い」人たちのものでなく，普通の人向けの「クール」で「ポップ」なものと見られるようになったしね。

とはいえ，こんな「こんなことができたら楽しい〜」的な企画案がゲームのバリエーションを広げることになるはずはない。これじゃゲーム自体完成できないし，したとしても世間に受け入れられるゲームになる可能性はゼロだ。

なぜか？

ゲームの本質への理解に基づいた，ゲームとしての軸がないからだ。軸のない企画は，固まらない。固まってないゲームは，プレイしてもぼやっとした印象しか残らない。これじゃいくら有名な人が作ろうと，いくらアートデザインがよかろうとダメだ。

その，ゲームの軸とは？

それは，すなわちゲームの勝ち負けの条件である。すべてのゲームは，プレイヤーが勝利を目指してプレイすることを念頭に置いてデザインされなければならない。ということは，勝ちと負けに至る筋道をデザインすることこそ，ゲームデザインの基本だということだ。

平凡な答えと思っただろうか？　が，勝ち負けに対する配慮をちょっとでもおろそかにすると，たちまちゲームシステムは破綻する。そして，ゲームが「クール」で「ポップ」だと思われている昨今，勝ち負けにこだわることは，とてもないがしろにされやすいのだ。「パラッパ」や「I.Q.」にもしっかりとしたゲームデザインがされていることを見落としては，成功はありえない。

よく思い出してほしい。あなたが面白いと思ったゲームは，例外なく「負けて悔しい，もう1回！」と思った経験があるはずだ。そう思わせることこそ，ゲームデザインの基本であり究極である。

自分のアイデアをゲームにするときには，まずプレイヤーにとってなにが勝ちでなにが負けかをしっかりと文章にしよう。これがゲームデザインの第一歩だ。ここから，それを軸としてルールを組み立てていく。

「パラッパラッパー」なら「うまくラップができるとステージクリア，ヘタだとゲームオーバー」だし，「I.Q.」なら「迫るキューブのすべてを沈めるとクリア，プレイヤーかキューブがステージから出てしまうか，キューブの下敷きになるとゲームオーバー」だ。もっと細かく描写することはできるが，最初から決めすぎると修正がきかなくなる。

自分のゲームのアイデアだと，どうなると勝ちで，どうなると負けなのだろうか？

うまくいい表せないなら，まだゲームとしてのイメージができていないことになる。特にシナリオ中心にアイデアを考えているときにはありがちだ。ひょっとしたらそのアイデアは映画の脚本にしたほうが生きるかもしれない。この辺は冷静に見極めよう。他人に話してみるのもいいだろう。

マクロからミクロへの帰納

勝ち負けの条件は文章になっただろうか？

平凡なアイデアに見えるかもしれないが，ここでガッカリする必要はない。ゲームの概要はゲームオーバーとクリアだけでは決まらない。

たとえば「バーチャファイター3」。「敵との3本勝負に勝ち，DURALを倒すとゲームクリア。3本コンピュータに取られるとゲームオーバー」だ。

が，それをいちばん大きな枠として，そのなかに「ひとつの試合に勝つ/負ける」，さらに「ダメージを与える/受ける」という勝負が入れ子状に存在している。プレイヤーは，局面局面の勝利/敗北を繰り返しながら最終的な勝利条件への到達を目指す。

だから，ゲームオーバーとゴールは同じでも，局面のルールが違えば違うゲーム性を持ったゲームになる。たとえば，格闘ゲームで，1本目から3本目になるにつれて，ダメージの大きさが変わるようにしたら？　あるいは体力を次の勝負に持ち越すようにしたら？

自分のアイデアも同様に，細かいところに落としていこう。操作方法とか，メニューとか，イメージできるところは全部書き出してみる。そのときは，ゲームのなかで役割を果たす主体，それらの行動，状態（パラメータ）などを書いていく。人に見せる以前なので，箇条書きでもなんでも，書式はどうでもいい。形にこだわると書きづらくなるので，メモみたいな形がおススメだ。

イヤがらずに，各レベルでの勝敗をしっかりイメージしよう。やればやるほどあとの作業は楽になる。これは特に，チームでゲームを作るときに非常に大事だ。だいいち，細かいルール部分には自分のアイデアを盛り込む場所がたくさんある。これをつまらないと感じては，デザイナーにはなれない。

What are my choices ?

さて，それぞれの勝負を考えていくときには，どうやってプレイヤーが勝つのか，その手段を切り離して考えることはできない。弾を撃つ，モノを買う，お金を貯める，ゲームによってイロイロあるわけだが。それらのプレイヤーができることと，その手段が生む結果（＝勝ち負け）は，同時並行でイメージを固めていく必要がある。

さて，その前に，勝利の快感について考えてみよう。なぜ勝つことは楽しいのだろうか？

もし100人中100人がクリアできるゲームがあったとしたら？

こりゃ面白くない。スリルがないし，失敗しない勝利なんて，他人にイバれない。逆に成功の見込みがまったくないと思える敵に何度も挑むのも，すぐにアキてしまう。やるだけムダだからね。

ここに勝利の楽しさの理由がある。「失敗の可能性があったところを，今回自分は成功した」という一種の優越感，これが勝利の蜜が甘い理由だ。前できなかったことができるようになった，ほかの人ができなかった壁を自分が乗り越えた，ということである。

プレイヤーができること，勝利への手段を考えるときは，常にその手段に失敗の可能性を持たせなければならない。プレイヤーの前にいくつか選択肢があって，そしてそれぞれに一長一短があるのがいい。これでプレイヤーは悩む。悩めば悩むほど，その選択があっていたときの嬉しさは倍増するし，失敗したときに次の選択肢を試したい気持ちが強くなる。

ジャンケンを例に考えてみよう。プレゼントをかけたジャンケン勝負がある。あと1勝すればプレイヤーの勝利。

さぁ，これで勝てればもちろん嬉しいだろう。プレゼントが手に入るだけでなく，「失敗の可能性をかわして，勝利した」と実感できるからだ。これ自体でもゲーム性はある。

ただし，これでは完全に運任せだ。たとえば，ここで，相手が「次はパーだ」と予告したら？　話は心理ゲームになり，選択肢の悩みはより複雑になってくる。

相手がいっているのはウソかもしれない。いままで相手はウソをついてきたか？　相手は予告することで，自分がどう有利になると考えているのか？　それとも勝つ気がないのか？　考えるべきことはグッと増える。

さらに，実は今回ジャンケンのあいこが両者負けになるルールだったとしたら？ こうするとパーで様子を見ることができなくなる。自分の選択肢としてグーがなくなり，相手はそれを計算できるわけだから，パーが候補から落ちて……。自分でも考えてみてほしい。まぁ，これをどうゲームの演出に組み込むかはちょっと思いつかないが。

ともかく，どのレベルの勝負であっても，常にプレイヤーには選択肢を与えておくこと。そして，その選択肢には有利なものを作らないことを心がけてほしい。格闘ゲームは「中段攻撃は下段ガードに対して有効だが上段ガードに対して無効」，RPGは「魔法使いは魔法が使えるが体力がない」という具合に，常にプレイヤーに選択肢と有利/不利を与えている。

迷路や地形の設計のときにも，常に複数のルートがあるように心がけよう。プレイヤーは常に迷いたいのである。迷えば迷うほど，「自分の決断ができた」と感じられるし，そのほうがゲームの結果を受け入れやすい。

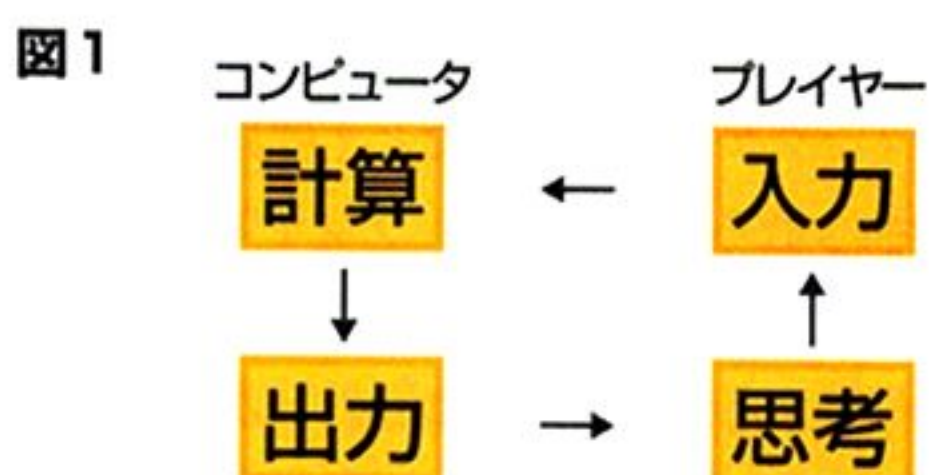
プレイのプロセス

さて，勝ち負けとはプレイの結果，そして勝利への手段はプレイのバリエーションのことだ。バリエーションと，それぞれの結果がリンクして動き出せば，ゲームのルールの構成は姿を現し始める。次に必要になるのが，チェックだ。

ゲームデザインとは，決定と変更の繰り返しだ。ひとつの案がまとまるまでの間にも，何回もルールは変更される。アイデアを思いつくことと，それをチェックすることが同時になるのも珍しくない。思いつくたびにゲームを作ってプレイしていては間にあわない。ゲームデザイナーはアタマの中でルールに抜けがないか，シミュレートできなければいけないのだ。

大変に面倒くさい作業だが，少しでも明確化し，考えを楽にするために，プレイの過程を次のように分析してみるといい。

基本的にはゲームプレイは図1のようになる。プレイヤーがコンピュータに入力し，計算結果が出力される。それをもとに次の手を考えてまた入力する，という流れだ。

図1

より詳しく考えてみよう。プレイヤーはゲームをどう受け止め，なにを考えて，次の行動を起こすのだろうか？

ここでは，どんなジャンルでも通用するよう，図2のような考え方をとってみた。

図2

大まかな考え方なので，基本的にどんなマシンでもどんなジャンルでも適用できる。アクションゲームはプレイヤーの入力を待たずに事態が進行するのでユーザーの認知や推測が追いつかない場合があるが，コマンド入力方式のもの（アドベンチャーなど）はこのプロセスとゲームの進行がシンクロして進んでいく。

それぞれを見ていこう。

1．認知

プレイヤーが現在の自分の状況をつかむこと。まずは，勝負の情勢，そしてその勝利条件と自分の現状のパラメータの比較，ひいては自分にどんなパラメータがあるかというゲームのルールの正しい理解を指す。自分の目標まであとどれくらいあるかを知り，それをもとに選択肢を認知する。

いま自分がゲームに勝てそうかどうかという「大局」と，現在自分が直面している状況という「局面」の2つがある。将棋でいうと，順調に相手の王将を追いつめていたのに，いま王手飛車取りを食らっている，とか。

2．推測

認知した状況をもとに，自分にどんな選択肢があって，それぞれの結果がどうなりそうかを推測する。将棋でうんうんうなっている時間のことだ。

シューティングゲームでも，たとえば目の前でボスが弾を撒き散らしたときに，プレイヤーはボムを使って撃退するか，よけきるか，あるいは自分のビームで相殺するか，といった選択肢がそれぞれどんな結果になりそうかをごく一瞬で考えている。

3．決定

それぞれの選択肢の成りゆきを予測し，損得勘定をした結果，「こうする！」と決める。これがゲームのイチバン楽しいところだ。将棋ならパチン！ と駒を打つところ……ではなく，その前の「この手にする！」と決めた瞬間のところを指す。

4．入力

その結果を外に出す。ゲームに伝える。そういう部分だ。3.の「決定」と分けたのは，ある種のアクションゲームでは「入力しやすいかどうか」という手作業の部分が，ゲームデザインに大きく影響してくるからだ。確実に入力できないかもしれないというのはリスク要因として決定に影響を与える。リアルタイムシミュレーションでは，操作が間にあうかどうかが戦略に影響を与えている。

1～4をまとめ直すと，自分がゲームルールのなかでいまどういう状況にあるか理解し，選択肢を見てそれぞれの結果を想像し，そのうえで次に下すアクションを決定し，それを入力する，ということになる。

これがいっぱいあるゲームは，やっぱりヒットする。

プレイヤーがゲームに熱中しているときは，このループが滞りなく回転している状態，そして熱中できなかったり，「クソゲーだなぁ」と感じているときはこのループが壊れているわけだ。だからプレイヤーの思考を中断しないような，スムーズな進行のできるゲームデザインを考える必要がある。

そうはいっても，実践するのは容易ではない。ここではマイナスを消すチェック方法を伝授したい。あらゆるクソゲーは，このプレイヤーの思考のプロセスをどっかでジャマしているためにプレイヤーの不興を買っているのだ。

クソゲー症例分類の巻

1．認知不能

現状がどうなっているかわからない。また自分のパラメータがゲームの事態にどう影響を与えるのかわからない。わからなければ，プレイヤーにとって存在しないのと一緒だ。あなたが設定したルールの体系は，プレイヤーによって崩されてしまう。

2．推測不能

用意された選択肢がどういう結果になりそうか考えることができない。これではゲームの迷う楽しみを味わうことができない。選択肢の意味が伝わっていない場合だ。また，選択肢が魅力に乏しかったり，有利と不利の条件提示がしっかりできていないものなども推測不能の病名になる。

なお，ライトユーザーには推測の力が弱いので，間口の広いゲームを作ろうと思ったら，ここがわかりやすくなるよう，手助けをしてやる必要がある。

3．決定不能

たいていの場合は決定不要だ。決められない，というより決める必要がない。選択肢のうちどちらかが決定的に有利だったり，決定的に不利だったりする。こうなったらゲームは単純な作業になってしまう。ゲームのいちばん楽しいはずのところがないわけだから，クソゲーと呼ばれるのは時間の問題。あとは，認知不能，推測不能になっていたら決定も不能になっていることはいうまでもない。

4．入力不能

これは，イチバンイメージしやすい。レスポンスが悪い，的確な入力ができない。入力方法が難しい。プログラムが原因になっていることが多いので，企画が紙のレベルではあまり問題になることはないかもしれない。

特に「ストリートファイター」シリーズはこの問題がデザインの大きな課題だ。最初はめったに出ない奥義だった昇竜拳も，いまのプレイヤーにとっては基本技。「出したいけど出るかどうか」という迷いがないので，投げ返しとか，ダウンキャンセルとか，いろいろ要素を増やしている。これはゲームを複雑にし，初心者には近づけないゲーム内容になってしまうおそれが多大にある。これは操作性をゲーム内容に絡めたデザインの場合に共通していえる問題だ。

そして，その先

こうやってチェックとプランを繰り返して，アイデアがゲームのルールとして蒸留され，定着していく。そのルールにはいろいろ長所短所あるかもしれない。だが，上にいったことが守られてさえいれば，ゲームとしての魅力を備えることは難しくない。

コンピュータに関する技術がいくら進んでも，いいゲームの本質は常に変わらない。すごいCGを見るためにあくせくするよりも，よくできたトランプゲームのほうが長く遊びたかったりするのは，そういうことだ。ゲームデザインの軸は，常にルールがエキサイティングであること。絵や音はあくまで肉なのだ。

そろそろCGのキレイさだけに頼ったようなゲームデザインは飽きられる頃だ。ゲームをする人は常に，自分で事態をコントロールしたいのだ。アナタがプレイヤーを押し込めるようなゲームデザインは避けなければならない。成功への道筋は多いほどよい。

より多くの人を引きつけ，海外も含めて幅広く評価されるためには，絵や音のクオリティ以上に，選択肢のクオリティ，勝利へ至るプロセスの「迷わせ方」にウデをふるってほしい。

ゲームの勝利の方程式の解は，ひとつ以上でなければならない。ひとつの解答で，ゴールへの図式がずらずらっと解けてしまうのでは，迷いがない。その迷いをいかに楽しく提供できるかが，ゲームデザイナーの仕事なのだ。

最後に

ゲーム作りのコツなんて，人それぞれというか，自分なりの理論さえあればそれでいいものかもしれない。あるいは理論さえ使わずカンで作る人もいるだろう。それでも，自然とその人なりの「筋」はあるはずだ。そして，お互いの持っている「筋」を披露しあって切磋琢磨することには，意味があると僕は思う。

アナタがこれを読んで，アイデアだけで完成しないモンモンとした状態を抜けられたりしてくれるのが，僕の願いだ。ひとりでもゲームを正しく理解したデザイナーが出てきてほしいと思う。

ゲームはプレイヤーの入力に応じて状況が変化していくメディアだ。逆にいえばプレイヤーの入力がないとゲーム内容のよさは1%も引き出されない。映画みたいに難解でもつまんなくても黙って座っていればすべての内容が見られるメディアとは違う。「ガマンして見てたら後半面白くなってきた」というのは，ゲームにはありえない。とにかくユーザーへのサービスが大事な，ある意味映画の演出より難しい部分のあるメディアなのである。

あなたも自分の好きなゲーム，あるは好みではなくても，よくできているヒットゲームを見て，ここで話した要素に分解して見るクセをつけてみよう。きっと，いままでにないゲームの見方を発見するだろう。

今後，この原稿を否定したり，追加してくれる人間が登場してくれることを願う。では。

近代縦射芸特講

八重垣那智

特殊な市場構造を持つアーケードゲーム業界。そこでのゲームの原点ともいうべきものが縦スクロールタイプのシューティングゲームだ。それは業界の縮図として多くの問題を内包しつつ，常に生き残ってきた。厳しい時代に立ち向かっていく姿勢に学ぶものは多いはずだ。

いまやビデオゲームというと，その名で呼ばれるものは，内容や形態のどれを取っても多種多彩でキリがない。マスコミに流れる情報，繰り返されるCMは，老若男女を問わず一般家庭の中にまでビデオゲームを滲透させている。ビデオゲームは，きわめて一部の実験的なものを除けば，作り手であるメーカーによって供給され，プレイヤーが対価を払うという商業的な構造の下に繁栄している。正確には商品として生み出され，市場が成立しているものだということだ。

そうしたビデオゲームにおける市場を最初に構築し，いまもって忠実に維持しているのがアーケードゲームの世界である。いわゆるゲームセンターにおけるビデオゲームであり，20年以上にわたり原則的に100円1プレイというスタイルを踏襲している。プレイヤーが望むだけ，何度でも対価を支払ってプレイできるという，内容と対価が比例する点が独特だ。このきわめて残酷な娯楽と対価の関係は，その黎明期から現在まで微塵も崩れてはいない。

ここでは，そのアーケードゲームで，もっとも綿々と作法が受け継がれ続けてきた，不滅のジャンルである「縦(画面/スクロール)シューティングゲーム(以後，縦シューと記す)」に着目し，その総括を試みたい。

縦シューを現状から理解することは，それらが経験してきた歴史や流行の蓄積を実例として捉えることにほかならない。縦シューはいつもアーケードゲームとともにあり，過ぎた時間の中にさえも道標として残っている。これらの過去から現在へのベクトルを捉えることは，未来の縦シューの姿，ひいては未来のアーケードゲームの姿を導く指標となりうるはずだ。

そこでまず，現在のアーケードゲームという世界（市場）を理解し，そこに置かれた現代の縦シューの生の姿を認識するところから始めていくことにしよう。

アーケードゲームの世界

アーケードゲームは，最初に構築されたビデオゲームの市場としては，実際にきわめて特殊な形態を持っている。プレイヤーの欲求（評価）に応じたプレイの反復により，支払われる対価が比例する，内容と対価の評価が残酷だと最初に書いたが，これはアーケードゲーム市場の一部分しか捉えていない。

その理由は簡単だ。

まず，メーカーの商品を買うという，本来の意味でのユーザーにあたる存在は「ゲームセンターの経営者」である点を確認しておこう。プレイヤーはあくまでもゲームセンターを訪れ，ゲームセンターが提供する「遊び」に対して対価を払う存在なのだ。アーケードゲームを生み出しているのはメーカーだが，それをプレイヤーに実際に提供しているのはゲームセンターである。この構造は，メーカーと提供者，ユーザーとプレイヤーがそれぞれ一致している家庭用ゲーム機の市場とは，大きく異なっている。ゲームセンターでのプレイヤーは，あくまでもゲームセンターにとっての客であり，メーカーの客ではないのである。

メーカーから商品がプレイヤーに渡るわけではないし，プレイヤーが支払った対価の多寡はメーカーには反映されない。対価の流れは，それぞれ独立したものだ。このため，プレイヤーによる売り上げが経営者にとってのゲームへの評価になり，ゲームセンターへの販売枚数がメーカーにとってのゲームの評価になる。こう考えると，両者の意識の違いを改めて認識できる。対価に対する感覚や，ゲームに対する評価が，互いに別の次元で語られているのだ。これはアーケードゲームを理解するうえで，きわめて重要なポイントといえる。

そういう意識の違いを含めると，プレイヤーに人気があり，高い売り上げが続き，多くのゲームセンターに出回るようなゲームが，すべてにとって理想的で完璧なのではないことに気づく。ゲームセンターにとっては飽くことなく売り上げが続くゲーム，つまり新たな投資の不要なゲームが究極の理想であるが，メーカーはやはり次のゲームもゲームセンターに買ってもらわねば困ってしまうからだ。

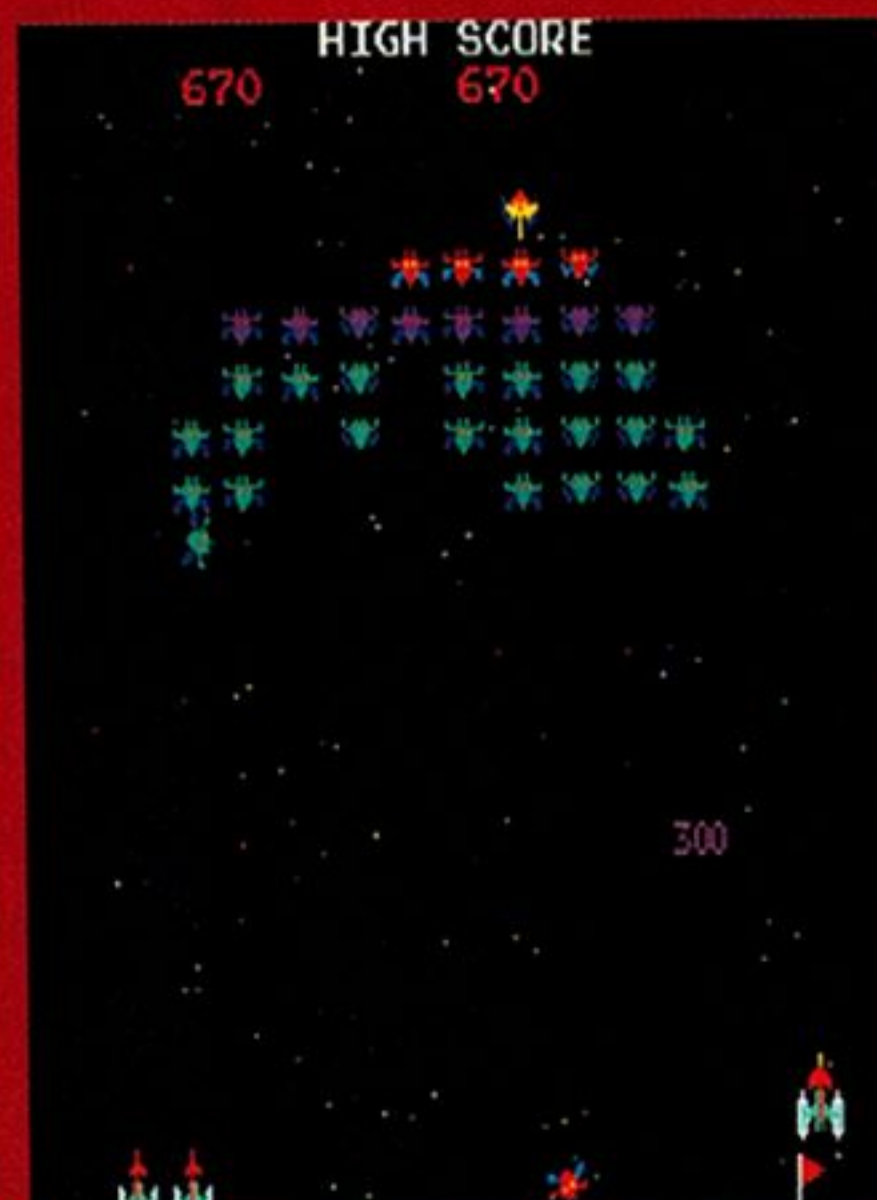
とはいえ，メーカーはあくまでもゲームセンターが欲するゲーム，ひいては何度もプレイされ続け，高い売り上げを出すゲームを目指して商品を作っている。それは誰にとっても理想的なゲームというものが現実的には不可能なことを，メーカーもゲームセンターも知っているからだ。

またゲームセンターでは，10人が100円ずつ遊ぶゲームよりも1人が2,000円遊ぶゲームを原則的によいゲームと評価する。どれだけ情報が溢れ，メーカーが宣伝し，多くのプレイヤーの注目を集めたとしても，ゲームセンターの現場では売り上げといった絶対評価に逆らうことはできない。商売だけに，最終的にこうした数字の力に振り回される世界であることも，同時に理解しておくべきことだ。

いまに生き残る縦シュー

このように複雑な構造を持つアーケードゲーム市場において，縦シューはどんな地位を得ているのだろうか。その現状を確認してみよう。

実際にゲームセンターに行くとわかるのだが，ゲームセンターにおける縦シューの台数比率は，おせじにも高いとはいえない。台数の少ないゲームセンターでは，その存在が皆無であることさえも驚くことではないだろう。もしあっても4～5台がいいところであり，しかも2年ぐらい前の旧機種がその中に平然と置いてあることも珍しくはない。ハッキリいうと，縦シューは不遇をかこっているのである。

スペースインベーダー（タイトー'78）
インベーダーは，海賊版以外にも正当な続編がある。ただ単なるリメイク的価値が強く，だんだん古く感じてしまうのは残念だった

ギャラクシアン（ナムコ'79）
敵の点数が，条件で変化するというのは，当時としては斬新。しかし点を稼ぐより長く遊ぶほうが，優先された時代でもあった

ムーンクレスタ（ニチブツ'80）
合体によるパワーアップは，慣れれば失敗はしなくなるもの。ほかにもこのゲームは敵が一切弾を射たないなど，意外な特徴もある

ギャラガ（ナムコ'81）
ボーナスステージといった，現在は廃れてしまったような作法なども，このゲームで確立されている。ある意味で永遠の名作かも

理由は，冷静に考えてみれば明白だ。それは売り上げの数字，プレイに必要な時間がその答えである。現在のゲームセンターの中心である対戦格闘ゲームと比べれば，そこにプレイヤーが満足するために必要な，時間に対する感覚の大きな違いを発見することができる。

対戦格闘ゲームの場合，いまや当然となっている乱入対戦台で人間同士が対戦する場合，3本勝負のプレイ時間は1本あたりが1分としても最大3分だ。どちらが勝っても，プレイヤーの巧拙にほとんど依存せずに，ほぼ1時間で20コインのペースで売り上げが生まれることになる。対して縦シューの場合，3分というプレイ時間では，満足を与えることさえ難しい。個人差もあるだろうが最低でも5分，できれば10分程度の時間遊べないと，満足したという言葉は出てこないものだ。しかし，1プレイ5分と考えても1時間で12コイン，プレイヤーが上達すれば必然的にプレイ時間が延びるため，むしろその数字は減少する傾向を否定できない。

これはプレイヤーの満足が，両者のゲームの構造の相違に依存していることを表している。格闘ゲームの満足は限られた時間内での，結果の良否に対して得られるものであり，縦シューの満足は，プレイが続くことによる快感を繰り返すことで得られるからだ。結局のところ縦シューというゲームシステムは，売り上げにおいて理論的に限界を上げることが難しいのである。

column *縦シューの定義*

ここでは縦シューを以下の定義により定めています。

①画面は縦画面配置であること
②自動スクロール，もしくは固定画面であること
③自機の移動操作に連動した任意方向への射撃ができないこと

こうした理論は，改めて証明するまでもない。自分が縦シューをプレイしたとき，どの程度の時間をプレイすれば満足するのか考えたり，ゲームセンターの現場で売り上げの評判を調べてみれば明白だ。だからこそゲームセンターにおける受難の時代は，縦シューから離れることなく続いているのである。

縦シューが守ってきたもの

このように不遇をかこっている縦シューであるが，こうした売り上げ至上の世界において，なぜか絶滅する兆候だけは見られない。少ないとはいえ，一定のペースで新製品もリリースされ，ときおり話題にもなっている。過去に話題や売り上げで席巻したゲームの中には，いまや何年も類似のものさえ作られなくなったジャンルのものがいくつもあるにもかかわらずだ。縦シューが，そうした消えたジャンルとならないのはなぜなのだろうか。

理由はいくつか考えられるが，もしもっとも大きな理由があるとすれば，それは縦シューがゲームの原点として，無意識に認知されているからであろう。現代の縦シューは，初期のゲームの時代から，延々と生き延びてきているがゆえに，その歴史的蓄積による概念，試行錯誤による熟成といったものに対して，他ジャンルの追随を許さないものになっている。継続は力なり，というべき領域に到達した希有なゲームジャンル，それが縦シューなのだ。いまなお最盛期の勢いと，精力的な新機軸化が進む対戦格闘ゲームでさえ，縦シューが獲得した数々の概念や熟成には遠く及ばない。

そこで改めてこの論の本題として，縦シューに蓄積された歴史の澱を見極めていくことにしよう。なぜ縦シューがゲームの原点として認知されているのか，つまりどうしてここまで縦シューが生き延びてこれたのか。そうした疑問には，縦シューの過去や歴史が，必ずや雄弁に真実を語ってくれることだろう。

古代から近世へ

縦シューが歩んできた歴史は，何度も書いているが，ビデオゲームやアーケードゲームが歩んできた歴史でもある。ゲームが電子技術の発達などと同期して，その規模や表現能力を急速に発達させてきたことは，説明無用の事実だろう。それを頭に入れると，黎明期のゲームを現在と比した場合，規模や能力は明らかに稚拙で卑小なものといわざるをえない。だがしかしここで見極めるべきは，その当時に生み出されていまに伝わり，そして残っているゲームの純粋要素とでもいうべきものだ。だから最初に，古いゲームに対してのそうした偏見をここで捨ててしまわなくてはいけない。

そうしてまず過去を振り返るときに，真っ先に挙げられるタイトルは，スペースインベーダー（タイトー'78）だ。縦シューというものの一里塚と考えてもいいし，日本のアーケードゲームそのものの原点と考えるのもいい。それだけ偉大な作品なのだ。

この名作は縦シューそのものの概念を一度に生み出している。それは，数多くの敵に対し，自分は孤独であるという，多数対自分ひとりというゲームの基本構造のことである。この構造はこれ以降のゲームにおいて，縦シューという枠を問わず，ビデオゲーム全体に滲透した基本常識ともいえる概念だ。数多くの敵の攻撃を避け，自分ひとりで敵に立ち向かう。レバーで自機を動かして攻撃を避け，ボタンで敵を攻撃する弾を発射する。これがまさに縦シューの原点だ。そのすべてが，この作品だけで構築されたのである。

ポストインベーダーの世代では，ギャラクシアン（ナムコ'79）における，手順型のテクニカルボーナスの採用に注目したい。待機している敵と，攻撃している敵の得点が異なるなど，スコアシステムと呼びうるものが構築されているのが斬新だ。現在は，敵の得点が指定の手順を踏むことで高得点に変化するシステムは当たり前だが，このゲームにルーツがあることに気づいておきたい。

スコアと縦シューが，非常に密になっているのは，単なる腕前を競える理由だけではなく，こうしたプレイヤーの能力を露にするシステムがきわめて初期に構築されているからなのだ。

次に語られるべき要素はパワーアップだ。これはムーンクレスタ（日本物産'79）とギャラガ（ナムコ'81）に見られるような，パワーアップ行為そのものにリスクを背負ったものがその発端だ。攻撃力を増やすために，手順を踏んで自機同士を合体させるという共通点を持っているが，つまりは弾を射ちやすくするために自機が大きくなるということだ。自機のシステムの中で，攻撃のメリットと回避するリスクが完結している。こうした背反的な仕組みが初期のパワーアップにおける特徴だ。

そして遂に縦シューを縦シューたらしめるスクロールの概念がゲームの世界に現れる。ただし当初導入された概念では，地面と高さを表現しようとする意識が強かったことが特記事項だ。ミッションX（データイースト'81）やゼビウス（ナムコ'82）などは，地上と空中の攻撃が別途になっている，典型的な高度を意識した縦シューだ。特にゼビウスは，その後，数多くの縦シューが，空中と地上を2つの武器で射ち分けるスタイルを踏襲するなど，その影響力の大きさに驚嘆すべき存在だ。いまから考えても，間違いなく，初期縦シューの完成形と断言できる名作である。

これら初期縦シューは，急速な技術の変化に晒され，それぞれの形態は多種多様すぎて，全体的には漠然としている。ただ，そうしたなかでもこれらには，絶対的に守られている大原則がある点を忘れずに押さえておこう。

それは，必ず敵を狙い射って倒さねばならないというシューティングゲームの原点だ。弾を当てるために敵の正面に構え，ボタンを押して攻撃する。弾は避けなければならないが，危険な敵の正面で対峙する。これこそが当時の縦シューの醍醐味であったといっていいだろう。そして，一見完成したかに見えた縦シューは，革命の時代を迎えていく。

革命の始まり

現在，縦シューの革命と呼べるようなものは，実際の革命とは異なり，多くの縦シューが出続けることで徐々に進行したものだ。特定のタイトルを境に縦シューというものの概念が激変したというのではない。その最初の兆候は，誰も気づかないような些細なものから始まっていたのである。

最初の兆候が見られたのは，一見単なるギャラガ（ナムコ'81）の続編といわれていたギャプラス（ナムコ'84）と考えられる。これは自機のシステムの中に，リスクを伴わないパワーアップ要素が存在している初のゲームだ。ゲームが進行していくうえで，より圧倒的な敵を倒すためにパワーアップを行う。もといパワーアップが必要になるという，ゲーム全体的なリスクが意識されている。従来のパワーアップの概念である攻撃を避けにくくなるリスクや，時間制限があるようなものが混ざっており，一見すると新しさを見分けにくいが，パワーアップに対して新しい世界を切り開いた存在であることは間違いない。ここにきてパワーアップは，よりゲーム全体を楽しむための前提として，縦シューの世界そのものに溶け込んだのである。

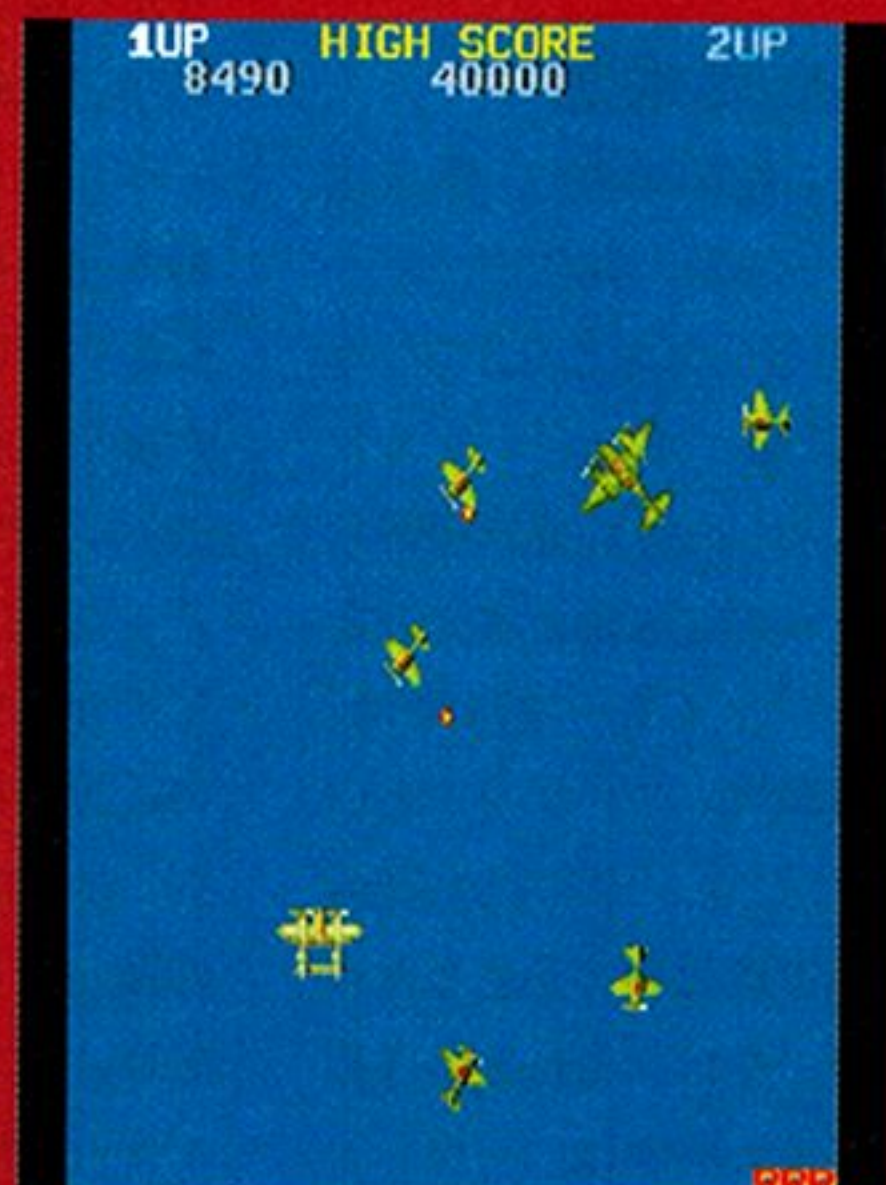
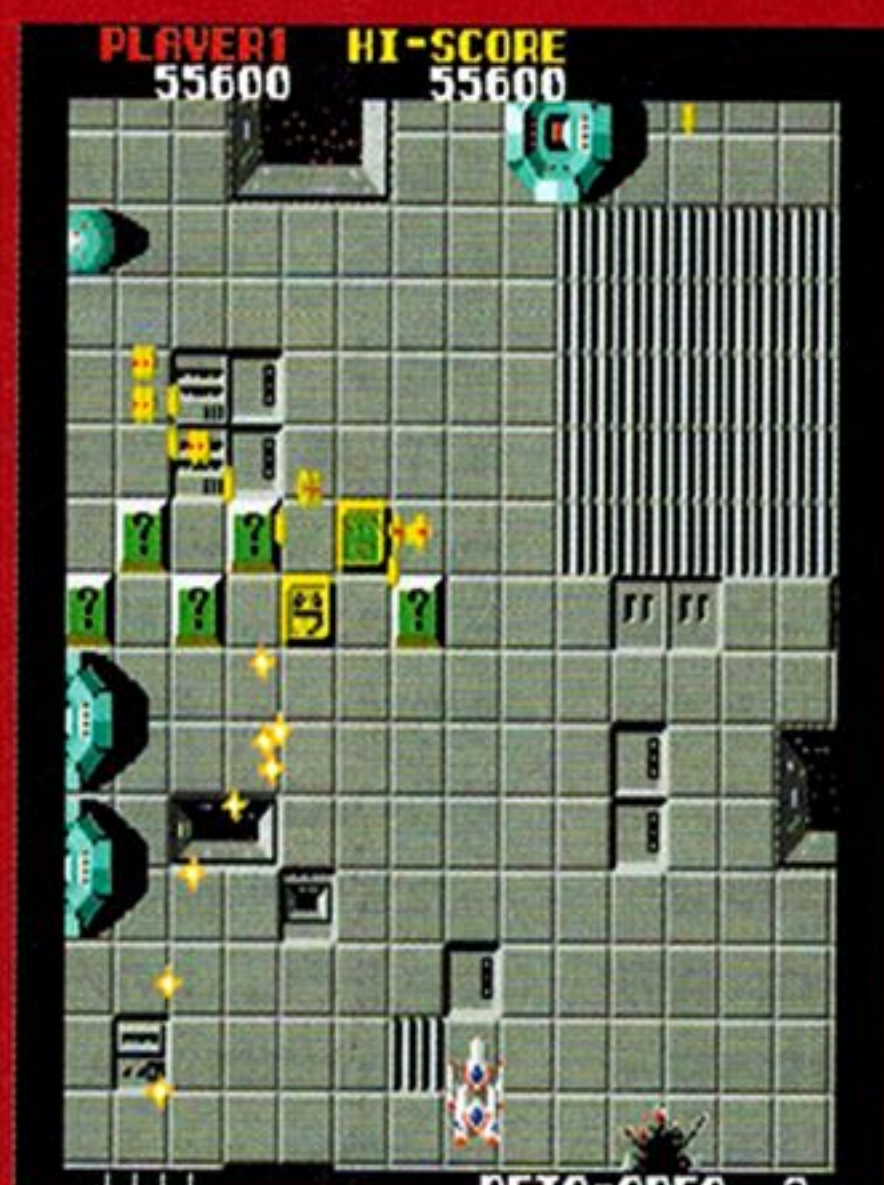
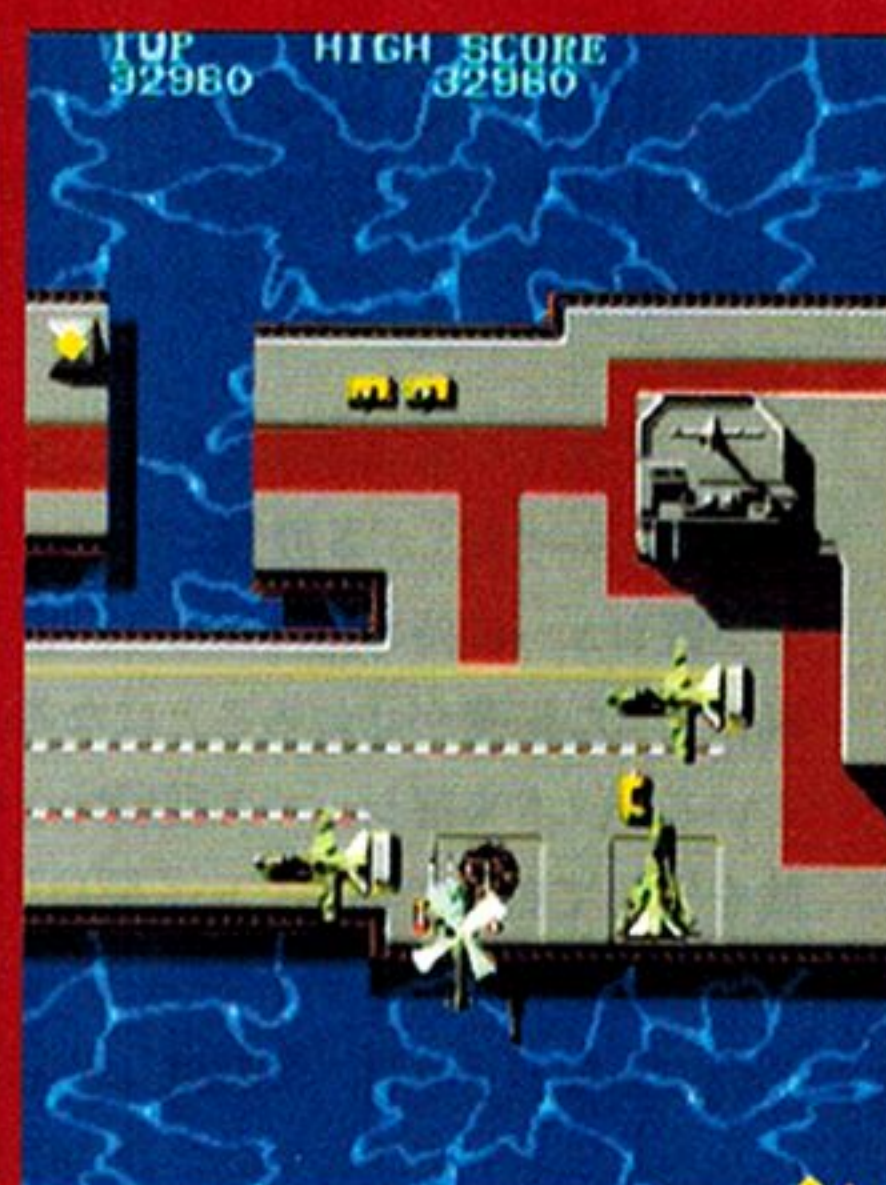
そして次に縦シューを変えたものとして挙げられるのは，敵の固さという，いまや当たり前の概念である。これはエクセリオン（ジャレコ'83）やバルガス（カプコン'84）がヒットしたことで，急激に広く認知されるようになったものだ。固い敵は強い敵。固さはその程度によって，敵の強さを明確に定量化することができる。これによって縦シューが表現できる領域を，大きく広げることに成功したのだ。そしてこの概念は，プレイヤーに連射という新しく高度な遊び方を，自然に要求していくことになる。

さらに1942（カプコン'84）では，巨大で固い敵，すなわちボス敵といったものの説得力を大幅に進歩させている。もちろんゼビウス（ナムコ'82）ですでに大型のボス敵と呼べる存在は出現している。しかし大きさと強さをあわせ持つことで，その存在をアピールできるようになったのは，固さの概念による表現がなによりも大きいといえる。

そしてその強さの象徴的な存在を，よりよくゲームの展開に取り入れていったのが，B-ウィング（データイースト'84）やスターフォース（テーカン'84）である。この2つのゲームは，プレイの単位としてステージという構造を持ち，その最後に必ずボス敵が登場する。ボスの破壊＝ステージクリアという概念は，この頃に確立されたものなのだ。

また1942（カプコン'84）は，ゲーム全体の長さを意識させた初めての縦シューでもある。腕さえあれば，永久に遊べることが当たり前だった当時，縦シューにおいて全面クリアという概念を導入した先見性は，注目するに値するものだ。さらにスターフォース（テーカン'84）は，空中と地上の敵を，明確に区別しながらもひとつのショットボタンだけで一様に攻撃できる，というシステムを採用した最初の縦シューとして名高い。同様にB-ウィング（データイースト'84）には，パワーアップによって自機から発射される弾が根本的に変化し，場面や状況で使い分けられるシステムが採用されている。いまでいう武器チェンジの思想が，すでに実現されている点は，見逃すことはできないだろう。

こうした小さな変革が積み重ねられていくうちに，徐々に縦シューには戦略や戦術が生まれていった。これらに限らず数多くのゲームによって，少しずつ縦シューは変化を実現し，それを土台に内容の奥深さや厚みというものを手に入れていっ

ゼビウス（ナムコ'82）
隠しキャラの元祖のといわれるが，どちらかというとグラフィック系の価値の高いゲーム。光源が明確に意識されている画像表現に注目

1942（カプコン'84）
射程距離があったり，いまと比べるとずいぶん窮屈。残りステージが意識されているとはいえ，全30ステージ以上の長丁場。先は長い

スターフォース
地上物に弾を遮られ，空中物が倒せないジレンマの概念がある。ただ地上と空中は独立して敵が出現する。配置の妙は偶然に依存だ

タイガーヘリ（タイトー'85）
ボンバーの元祖は緊急回避性格が強い。上級者には不要，初心者には使いこなせないといわれ，存在価値を疑問視する見解もあった

究極タイガー（タイトー'87）
近代縦シューを産み出した完成度に注目したい。実は海外版は2人同時プレイ。国内にもこれが出荷されてたら歴史は変わったかも？

ドラゴンスピリット（ナムコ'87）
毎面別世界，毎面ボスなどは近代化の兆候があるが，終盤のコンティニューで一定の面まで戻されるなど，やや理不尽な設定も目立つ

イメージファイト（アイレム'88）
独自の位置を築いている，タイトロープ縦シュー。続編がなく，類似品が僅かに認められる。システム的には，かなりレアな代物だ

雷電（セイブ'90）
文句なしの近代シューティングの原点。ただこのゲームが受けた要素を分析せずに，上辺の難しさを引継いだ類似品には，ちと閉口

たのである。

近代化への道

さらに進む縦シューの変化は，遂に頂点を迎える。それはボンバーの登場だ。通常のショットボタンに添えられた，もうひとつの攻撃兵器。強力だが，しかし使用回数には制限がある。いまや縦シューの必須要素として疑う余地のない地位を確立している。これが初めて登場したゲームはタイガーヘリ（タイトー'85）だ。とはいえボンバーは，当初は緊急用の逃げ道的な要素が強く，ゲームの構成上で重視されたものとはいいがたいレベルであった。それまでにも見受けられた，時間で制限された特殊兵器の延長線上に捉えられ，そのデビュー当時は意外にも地味な存在にすぎなかったのである。

むしろその同じ頃に縦シューの世界が手に入れたもののほうが，より明確に縦シューの世界に滲透している。それは2人同時プレイだ。交互にひとりずつプレイすることが当たり前だったゲームにおいて，同時に2人が協力してプレイするという要素は外見的にも大きな変化であった。しかもツインビー（コナミ'85），エグゼドエグゼス（カプコン'85），ハル21（SNK'85）と，3機種が同時期に集中して登場したために，2人同時プレイという概念は，高いインパクトをもって迎えられたのである。

そして変化の最終要素として，ハレーズコメット（タイトー'86）が多段階パワーアップというモノを縦シューに持ち込んだとき，縦シューの大原則に大きな転機が訪れることになる。先にも述べているが，本来縦シューの醍醐味というのは，危険を冒して狙って射ちにいく駆け引きにあった。しかしハレーズコメット（タイトー'86）は，強力で画面を覆うような，広範囲に及ぶ攻撃をパワーアップによって与えている。だがそれにより，敵を圧倒する爽快感という新しい感覚を縦シューの醍醐味に加えた初のゲームとなった。ゲームシステム的に敵を逃すことを悪としたことで，敵を殲滅することの爽快感がより強調された点も見逃せない。このときから縦シューは，強力なパワーアップで敵を掃射し，全滅させるためのゲームとして生まれ変わったといっていい。あえてこのゲームを近代縦シューの祖と位置づけても，決して的外れではないだろう。

こうして縦シューが近代化していくための要素は，形のうえでは出揃ったことになるわけだが，これは，現在から振り返ったからこそいえる結果論にすぎない。当時は変革と試行錯誤の真っ只中にあり，可能性を秘めた新しい要素が次々と試されていたのである。ダーウィン4079（データイースト'86）における進化と呼ばれる特殊なパワーアップシステムや，バミューダトライアングル（SNK'87）における特殊レバーによる砲塔システム，ジェミニウィング（テクモ'87）でのアイテム化されたパワーアップ攻撃など，例を挙げればきりがない。

結局，近代縦シューというものの完成を見るためには，変革の時代に生まれた要素を選別し，バランスよく吸収するための時間が，なによりも必要だったのである。

たどり着いた完成形

そうした熟成期ともいえる時間の中で，際立っていたのは飛翔鮫（タイトー'87），究極タイガー（タイトー'87）と連続してリリースされた2つの作品だ。地上と空中の同時攻撃，ボンバーシステム，固い強敵など，数多くの当たりの要素を盛り込んでおり，当時は縦シューの完成形とさえ考えられていた。究極タイガー（タイトー'87）では，複数の武器の使い分けと多段階パワーアップといったものが完全に融合しており，10年以上経ったいまでも，その内容の充実度は衰えを感じさせないほど素晴らしいものだ。やはりこれも名作といっていいだろう。そしてその内容は，タツジン（タイトー'88）に踏襲され，さらに充実度を重ねていることでも確かであることがわかる。

とはいえこうした時期にも，当然ながら新しい縦シューのアイデアや要素といったものへのアプローチは続いていた。ファンタジックな世界観と，目まぐるしく変化するステージ。それらによってストーリー性を強く押し出したドラゴンスピリット（ナムコ'87）やフェリオス（ナムコ'89）などは，印象に残る縦シューの筆頭に挙げられるだろう。連射というプレイスタイルをゲームそのものが取り込み，初めからボタンを押したままで連射できるオメガファイター（UPL'89）や，ジオラマなどを取り込んで写実的な表現を盛り込んだツインイーグル（タイトー'88），複数の武器をボタンで切り替えるブラストオフ（ナムコ'89）なども，人によって価値の差こそあれ，記憶に留めるべき性格の縦シューといえる。

またそうしたなかでも，いまだにファンの多いイメージファイト（アイレム'88）は，この流れとは別個の特殊な存在としておきたい。接触するとミスになる地形。スピードをボタン操作で自在に変化可能な自機。レバー操作によって攻撃方向を同時にコントロールできる攻撃システム。これらの独創的な要素が強すぎることで，本来の縦シューから離れた存在として，単独で完結している例外中の例外と考えたほうがいいだろう。

そしてついに近代縦シューとしての完成形が登場する。それは雷電（セイブ'90）だ。奇しくもその5年前に，2人同時プレイの縦シューが3つ，その覇を競ったように，このときも，雷電（セイブ'90）にトライゴン（コナミ'90）とエアデュエル（アイレム'90）といった3つの縦シューが，ほぼ同時にその姿を現した。これらはいずれも，縦シューの近代化に必要な要素を吸収することに成功していたはずだった。当時，拮抗し伯仲したそれらの勝敗はプレイヤーの注目の的でもあった。この争いに，生き残り勝利したのは，もっともシンプルでゲーム性にもテクニカルな要素の少なかった雷電（セイブ'90）だった。近代縦シューの方向性は，この時点で大きく固定されることになる。

戦国エース（彩京'93）
選択したパイロットと自機の設定が絡むようになってきたのは，このあたりから。和風で，かつ荒唐無稽なのに意外にも違和感は少ない

実はその勝利の要因は，いまとなってもあまり明確ではない。むしろどうとでも取れるほど曖昧なものと考える人もいる。それでもあえて理由をつけるなら，それはゲームセンターにおける売り上げ至上主義の定着であったと思われる。

そのほかにもバブル経済の真っ只中であった社会背景も影響しているだろうし，ゲームセンターで遊びながら成長し，社会に出たサラリーマン世代が大人の娯楽としてゲームを嗜むようになっていたことも無関係ではないだろう。長年ゲームセンターで遊んできたゲームの楽しさを知っている大人たちは，子供や学生よりも売り上げに大きく貢献する存在となっていたのである。そうした状況において，誰もが単純なルールで楽しめる，徹底的によりシンプルに練り上げられた縦シューこそが，無意識に求められていた存在だったのではないか，と考えられるのである。

続編と異端の同居

こうして近代縦シューは完成形ともいうべき存在を手に入れたといえるのだが，それは雷電クローンとも呼ぶべき模倣が次々と生まれていく時代の始まりでもあった。対戦格闘ゲームの急速なブームにより縦シューというジャンルの地位が低下し，バブルの崩壊による景気の後退といった外部要因の中で，メーカーは直接の客であるゲームセンターが要望する縦シュー，平たくいうと雷電（セイブ'90）みたいな売り上げのものを望まれ，それに応えて雷電（セイブ'90）みたいな外見のものを生み出していたのである。

結局メーカーは，自分達の客であるゲームセンターが歓迎するゲームを作らなくてはいけない。安定して売り上げを望むゲームセンターが，そのゲーム内容に対して保守的な思想を持つのは断じて悪いことではない。必然的にメーカーが採る手法は，過去に評価の高かった作品が用いた手法を踏襲したものや，過去のヒット作品の続編というアプローチだ。雷電（セイブ'90）にさえ雷電II（セイブ'93）や雷電DX（セイブ'94），ライデンファイターズ（セイブ'96）といった続編が生まれており，とてもその例外には見えない。しかもこうした流れは，縦シューというジャンルに限った話ではない。ゲーム業界全体が，厳しい状況で生き残るために，こうした確かといわれる手法を率先して採っていたのである。このように縦シューばかりがお寒い状況に追い込まれていたのではなかったとはいえ，縦シューにはより強い向かい風が吹いていたことは忘れずに把握しておきたい。

しかも全体的な流れの中でも，縦シューの一部はゲーム本体とは関係ない演出面での特徴づけや，あえて異端ともいうべきシステムを一部に組み入れることで独自性を生む努力を行っていた。前者の代表はソニックウィングス（Vシステム'92）や戦国エース（彩京'93）であり，後者の代表にはグリッドシーカー（タイトー'92）やレイフォース（タイトー'93）といったものが並ぶ。これらは，縦シューの可能性とさらなる発展を目指して，諦めずに試行錯誤をした成果であると考えると納得できるだろう。

こうした試行錯誤の結果として，この頃から自機選択といった概念が定着し，プレイヤーの好みや選択による展開の違いといったものが縦シューの要素に加わっている。ひとつのゲームでより多くのプレイヤーを受け入れるような工夫や目に見える多元性が表現されるようになったのは，近代縦シューが新たに手に入れた，数少ない進歩した要素なのである。

またそのほかに，ポラックス（NTC'91）をはじめとし最近ではステッガー1（VISCO/AFEGA'98）に至る，日本製とは違った感覚を持った，それでいてスタイルや様式を踏襲した一連の輸入シューティングにも目を向けておきたい。技術的には同時代のものに比べて稚拙であり，その流通量も少ないために，ほとんど省みられていないレベルのゲームにすぎない。しかし，近代縦シューの原点であるシンプルさを頑なに守ろうとしていた点では，一定の存在価値と評価を与えるべきものと思われる。むしろ独自性を追い求めた結果なのか，あげく先祖返りしたようなゲームシステムを採用し，近代縦シューの爽快感やシンプルさを失ってしまったいくつかのゲームよりも，縦シューとしての完成度は高いものといえなくもないのだ。

商業主義の利点と限界

そして最後にいま現在の縦シュー実情に話が戻るわけだが，結局はいまでもアーケードゲーム全体が抱えてしまった，制作コストと市場規模の問題から，縦シューは逃れられていない。特に縦シューの製作コストでもっとも比重が高いの

レイフォース（タイトー'93）
テクニカルさゆえに敷居が高く見える部分で損をしている印象だ。演出面などカッコよく魅せることに終始しており，ファンも多い

スーパーX（NTC'94）
韓国製シューティングのひとつ。この頃は完成度もアップしている。気になるのは，部分的な演出や技術の綻び。ツメの甘さはやはり難

ストライカーズ1945（彩京'95）
現在を代表する縦シューシリーズのひとつ。シンプルでありながら，独創的であるのは難しいが，成功している点でその評価は高い

怒首領蜂（アトラス/ケイブ'97）
様式化された現在の縦シューの象徴というべきゲーム。幾何学模様のようなボスの連射，圧倒的な敵の攻撃，その難度ともども印象的だ

は，そのグラフィックである。なかでも広大で緻密な背景画像は，とても一朝一夕に描けるものではない。ハードウェアの進歩とあいまって，背景に限らずキャラクターのすべてに対し，職人技のような高度なレベルのものが当然のように要求されている。

そこで新しい技法として，3Dグラフィックレンダリングによる描画の技法や，テクスチャの実写からの取り込みといった手法が生まれている。ネビュラスレイ（ナムコ'94）などが有名だが，最近の縦シューでは，この技法は積極的に利用されている。高度な完成度を持つ画像を，より効率よく生み出すための強力な武器として，3Dグラフィックが縦シューを支えている。こうした技術的な現場でも，縦シューは生き残る術を探っており，決して矛盾や問題に対して，消極的ではないことは心強い事実だ。

とはいえ，あちこちで声高にいわれているように，近代縦シューという概念が確立してからというもの，革命的なゲーム要素の発見や登場は，従来に比べ極端に減少している。いままで振り返ってきたその歴史のなかでも，現在が進化の止まった停滞の時期であることは，否めない事実だ。

最初にも述べたように，売り上げのためにはプレイヤーに対し満足を与えるための時間を短くしなければならず，かといって難しくして飽きられるようなことがあっては自殺行為になるという二律背反の問題だ。このようにゲームに求められる姿は，むしろ過去よりも厳しく高度なものになっている。ただそれをクリアしようとして歩んでいく限り，決して縦シューは進化しなくなってしまったということにはならないだろう。

最新の縦シューをいくつか見てみると，いわゆる近代縦シューの基本に忠実な内容に，溜め射ちといった軽いテクニカル要素を添えた，ストライカーズ1945（彩京'95）のシリーズやライデンファイターズ（セイブ'96）のシリーズが，その代表として挙げられるはずだ。しかしその半面，バトルガレッガ（ライジング'96）や怒首領蜂（ケイブ'97）のような，一見，熾烈な画面や演出で派手さを演出し，そこに得点やパワーアップの緻密なシステムを盛り込むことで，高度な奥深さを実現したスタイルにも評価が高い。難しいという批判的な意見に負けず，好意的な評価を聞くことも多い。こうしたいわばマニアを意識したゲームが，それなりの評価を得られるようになってきたのは，意外にも面白い傾向だ。その意義や評価を，この時点で断じてしまうのはやや早計だろう。改めて振り返ることができるようになったとき，バトルガレッガ（ライジング'96）や怒首領蜂（ケイブ'97）が，次世代縦シューの転機として認められる日がくるかもしれないからだ。安直な評価ができないということは，まだまだ縦シューが新しいモノを求め続け，進化している証拠のひとつと受け取るのがいいだろう。

縦シューは死なない

こうして縦シューという世界の歴史を理解してみると，アーケードゲームの過去に埋もれていったその他のいくつものゲームジャンルとの違いは，もうすでに歴然としていることがわかる。

それはなにがあろうとも縦シューは新しいことを探し，進化しようとしてきたということだ。それはジャンルとして生き延びるためだけではなく，新しい楽しさを生み出そうとしてきた努力の歴史だ。その試みが常に成功し続けてきたといっても，もちろん間違いはない。何度も縦シューは行き詰まったといわれてきたし，それを否定できない時期もあった。しかし，所詮それは表面上の問題であり，結果としてきちんと生き残り，進化さえ遂げてきたのは，ジャンル自身が持つ力，バイタリティとも呼ぶべきものの賜だったことは，いまや疑う余地はない。

縦シューがこれからも変わり続けていく限り，アーケードゲームもきっと一緒に変わっていくだろう。面白いゲームがいくつも現れ，我々を楽しませてくれるはずである。いつか，縦シューの歩みが止まり，ゲームセンターから縦シューが消えてしまったとしたら，そのときはアーケードゲーム自身にも終わりがくるときだと私は考えている。裏を返せばアーケードゲームが永遠に，いつまでも続いていくとするなら，必ずそこには縦シューがあるはずだ。どこまでも，そしていつまでも。縦シューは決して死なないゲームなのである。

表1　'90年代業務用/縦画面/スクロール自動/上方攻撃基本シューティングゲーム一覧

阿修羅ブラスター	TAITO/VISCO?	'90
エアデュエル	Irem	'90
スカイスマッシャー	日本システム	'90
空牙 VIPER TRAIL	DECO	'90
トライゴン	KONAMI	'90
ドラゴンセイバー	NAMCO	'90
MJ-12	TAITO	'90
メタフォックス	SETA/JORDAN(ALLUMER?)	'90
雷電	TECMO/Seibu	'90
アクロバットミッション	UPL	'91
1941 -COUNTER ATTACK-	CAPCOM	'91
ヴィマナ	TOAPLAN	'91
S.T.G.	TECMO/Athena	'91
ガルフストーム	MEDIA商事/NTC?	'91
ガンフロンティア	TAITO	'91
サンダーブラスター	Irem	'91
ターボフォース	V-System	'91
出たな!!ツインビー	KONAMI	'91
ポラックス	ATLUS/NTC	'91
雷龍	TECMO/NMK	'91
F/A	NAMCO	'92
SDガンダム ネオバトリング	BANPRESTO/ALLUMER	'92
ガルメデス	VISCO	'92
ガンネイル	TECMO/NMK	'92
グリッドシーカー	TAITO	'92
蠍サンドスコーピオン	FACE	'92
ジンジンジップ	ALLUMER	'92
スカイアラート	METRO	'92
ソニックウィングス	V-System	'92
達人王	TOA-PLAN	'92
超時空要塞マクロス	BANPRESTO/NMK	'92
ドギューン	TOAPLAN	'92
バース	CAPCOM	'92
爆烈ブレイカー	KANEKO	'92
ファイナルスターフォース	TECMO	'92
アースジョーカー	VISCO	'93
ウォーオブエアロ	Allumer	'93
コズモギャングズ・ザ・ビデオ	NAMCO	'93
戦国エース	PSIKYO	'93
大王	Athena	'93
雙翼ダブルウィングス	MICHEL	'93
バツグン	TOAPLAN	'93
V.V（ブイファイヴ）	TOA-PLAN	'93
ブルーホーク	NTC	'93
ブレイガール2	HOT-B	'93
魔法大作戦	RAIZING	'93
雷電Ⅱ	SEiBu	'93
雷龍Ⅱ	NMK	'93
龍神	TAITO/EAST_Technorogy	'93
レイフォース	TAITO	'93
エイトフォース	TECMO	'94
ガンバード	PSIKYO	'94
疾風・魔法大作戦	RAZING	'94
スーパーX	NTC	'94
ツインイーグルⅡ	SETA	'94
ネビュラスレイ	NAMCO	'94
バツグンSPECIAL	TOAPLAN	'94
ファイヤーバレル	Irem	'94
マジンガーZ	BANPRESTO	'94
マッドシャーク	ALLUMER	'94
雷電DX	SEiBu	'94
ラピッドヒーロー	NMK	'94
あっかんべーだー	TAITO	'95
19XX -THE WAR AGAINST DESTINY-	CAPCOM	'95
ヴァリアメタル	EXERENT SYSTEM	'95
ウルトラ警備隊	BANPRESTO/SETA	'95
ゲーム天国	JALECO	'95
逆鱗弾	TAITO	'95
STRIKERS 1945	PSIKYO	'95
首領蜂	ATLUS/CAVE	'95
ナムコクラシックコレクション1	NAMCO	'95
バイパーフェイズ1	SEiBu	'95
バイパーフェイズ1(NEW)	SEiBu	'95
アクウギャレット	BANPRESTO/GAZZEL	'96
ガンドッグス	SEiBu	'96
究極タイガーⅡ	TAITO/TAKUMI	'96
スカルファング -空牙外伝-	DATA EAST	'96
ストームブレード	VISCO	'96
バトルガレッガ	RAIZING	'96
マクロス プラス	BANPRESTO/?	'96
紫炎龍	童	'97
怒首領蜂	ATLUS/CAVE	'97
ソニックウィングスLIMITED	V-SYSTEM	'97
閃激ストライカー	KANEKO/童	'97
STRIKERS 1945 Ⅱ	PSIKYO	'97
ライデンファイターズ2	SEiBu	'97
ArmedPlice バトライダー(A,B)	RAIZING	'98
サイバーン	KANEKO	'98
エスプレイド	CAVE	'98
ステッガー1	VISCO/AFEGA	'98
スペースボンバー(A,B)	PSIKYO	'98
ライデンファイターズJET	SEiBu	'98
ガンバード2	PSIKYO	'98

女一代ゲーム半生記

出口 香

ファミコンが発売されて13年あまり。その間ゲーム機はさまざまに勢力図を変えながら，一般家庭にじわじわと浸透していった。いまや一家に1台は当たり前といった状況で，昔からゲームに興じ，友達(もちろん女)から変人扱いされてきた私もようやく普通よりちょっとマニアだね，程度の感覚で受け入れられるようになった。いや〜，いい時代がやってきたもんだ。

でも，ちょっと待てよ!? 確かにゲームの数は昔に比べて格段に増えてきてるし，それなりに市民権も得ているみたいだけど，私個人としてはハマれるゲームとの出会いは昔よりむしろ減っているように感じるぞ。確かにいい作品もあるにはある。だがゲーム全体の数からいったらその割合は昔よりも確実に減ってきているように思えるのだ。

実際，私の中でいいゲームと思うものは，旧ゲーム機のものばかりだし。

まぁ，これらの理由はおいおい文中でにおわせていくとして，とりあえず私がハマったゲームをいくつか紹介してみよう。

黄金時代－ハマリゲーとの遭遇

最初にハマったのは「平安京エイリアン」。

かれこれ18年前のことだ。その頃は当然のことながらファミコンもなく，中坊だった私はバンドの帰りに渋谷のゲーセンで仲間数人とこのゲームに興じていた。まだスクロールがどうのキャラがどうのという時代じゃなく，ひたすらレバーとボタンを操作しエイリアンを捕まえるだけという単純なゲーム。しかし友達と一緒にプレイできるというのがなにより魅力的だったのだ。

そうこうする間にほかのゲームにも手を出し始め，ゲーセンのにーちゃんにクレジット20ぐらい入れてもらって「誰が最初に10本旗を出すか？」と「ギャラクシアン」に燃えまくったのだった。

それからさらに7年後。すっかりゲーセンからも足を洗い，健全な一少女としてマジメな日々を送っていた私に運命的な出合いが訪れた。

「ドラクエII」。

このゲームのために私はファミコンを購入し，その日から学生の本分も忘れて日がなゲームに没頭するようになったのだ。ひとり暮らしの部屋で画面に話しかけほくそえむ私。その姿を友人に見られてたら，すぐさま縁を切られていたろうな。

さながらジャンキー状態で「ドラクエII」にハマり続けた私は，ついにはその年，ファミコン雑誌の門戸を叩くまでに至ってしまった。まさに人生の転機を「ドラクエ」にゆだねた大バカモノ。

そんな私がファミコンでいちばんハマったのが「ドラクエIII」。ほかにも「ファミコン探偵倶楽部」とか「桃太郎伝説」とか「FF III」とか面白いものはあったけど，もともと「ドラクエ」バカだったんだもんでね。でもこの「ドラクエIII」は一種神懸かりと思えるほどデキがよかった。ストーリーはもちろんのこと，イベントの振り分けや敵のバランスも絶妙，ダンジョンにもかなり工夫が凝らされていた。もちろんゲームシステムもユーザーフレンドリー。かくして私の中で「RPG＝ドラクエ」という図式が確立したのだった。事実それ以後このタイプのRPGが乱発したしね。

それから数年，仕事に追われ，ゲームをやる時間も少なくなった頃，遅ればせながらメガCDユーザーとなった私はちょっと気になるゲームを見つけた。「ルナ〜SILVER STAR STORY〜」だ。「ドラクエ」が客観的に主人公を操作する感覚だったのに対し，これはプレイヤーがどっぷり主人公になりきるタイプのRPGだった。要所要所にビジュアルを挿入することでいやがおうにも盛り上がるシナリオ。だからといってイベント続きで当初の目的はなんだったっけ，みたいなこともない。経験値稼ぎをしなくてもサクサク進めるのも，RPGをやり尽くしていた私には新鮮だった。加えてBGMもかなりいいデキ。全体通してそうとう練られた作りなのに，どこか肩の力が抜けた感じに仕上がっているのがまたホッとさせてくれて。う〜ん，次回作が待ち遠しいぞって感じだった。

そして出ました，我が最愛の「ルナII〜ETERNAL BLUE〜」が！ ああ，メガCDユーザーでよかったよぉ。立ち上げ直後に流れるオープニング＆プロローグからして破格のデキ。アニメデモが多用されつつある当時のゲーム状況にあっても，ここまで力を入れたものはなかった。このプロローグは謎の少女ルーシアを十分すぎるほどに印象づけ，プレイヤーを引っ張る原動力となっていた。そして実際のゲーム本編でまたびっくり。なんと少女ルーシアは「謎」なだけに，プレイヤーの意思とは関係なく行動するAIキャラだったのだ！ しかもそのAI具合が巧妙で，彼女が仲間を信頼していないうちは逃げたり自己防御に終始するが，冒険が進み仲間との関わりを持つにつれ戦闘に参加していくようになるのだ。こりゃ1本取られたって感じだわ。

でもって，このゲームにはRPGお決まりの「お遣い」ってヤツがほとんどない。すべての行動はパーティメンバーに出会うためであり，結束を増すためのものなのだ。もちろん経験値稼ぎの必要はほとんどナシ。また各パーティキャラにも焦点を当てることで，さらに深みを持たせるといったストーリー作りのうまさ。はっきりいってここまで練られた作りのRPGはいまもってないね。

しかもこのゲーム，エンディングの作りまでもが巧妙なのだ。ルーシアが青き星に還っていき，主人公のヒイロが彼女を探しに出るシーンでエンディング，スタッフロールが流れ出す。ちょっと拍子抜けだなぁ，と思ってセーブデータを見ると，そこに見覚えのないデータが。

そう，これぞ世に有名な「二段落ちエンディング」。そのデータで旅を進めて数時間後，真のエンディングに出合えるという寸法。まさにプロローグあればエピローグありきというわけなのだ。もうすべてにおいて凝りまくりのこの「ルナII」。私の中では，いまだにこのゲームより上をいくゲームはない。

あれから4年，ようやくサターンにも「ルナ2」として移植されたが，こちらも前作のイメージを壊すことなくいいゲームに仕上がっている。それでもあの当時の驚きと喜びを思い出すと，やっぱりベスト2になっちゃうんだよなぁ。だって(真の)エンディングで泣いたゲームってあれが初めてだもん。「ルナII」との出合いが嬉しくて，ついには「ルナ」シリーズの公式設定資料集まで作らせてもらっちゃったし(あい変わらずハマるとなにをしでかすかわからない出口であった)。

青銅時代－クソゲーの誕生

そういや私にゃファミコンでさんざんクソゲーの名をほしいままにした「マインドシーカー」というヤツにハマったという過去もあったな。当時話題の超能力少年清田くんの監修による「超能力養成ソフト」と豪語するそれを，私は狂ったように毎日やり続けていた。マジで超能力が身につくと信じて！ 婚期目前の女がやるにはあまりにもイッちゃってるゲーム。たぶんその頃私は人生に疲れ果てていたんだろう，そう思いたい……。

おそらくこのゲームのエンディングを見た人間は，私を含めて世に100人といまい。しかしこのエンディングがもう超電波入っちゃってるシロモ

ノでねぇ。宇宙から新しい来訪者が(だったと思う)とか，人類は新しい時代に入る(これもだったと思う)だとか，確か数千年以上先のことまで予言してたぞ。そりゃそんだけ先なら宇宙人だってくるわな。そして極めつけに「君は僕の同志として，近々僕からのメッセージを受けるだろう」って。あれから10年以上経ついまも，私にゃそんなもの受け取った記憶はさらさらないが。

ただ，ひと言ハマった人間からいわせてもらえば，結果的に「マインドシーカー」はクソゲーになったとはいえ，悪意があったわけじゃなかった。ナムコの人々，特に清田くんの周囲で開発に携わっていた人は，ある種「これは革命的なソフト」だと意欲満々だったに違いない。事実，革命的ではあった。だがあまりに偶然性が強すぎるゲーム性にプレイヤーはついていけなかったのだ。

こういった制作意欲は満々なのに「ちょっとした勘違い(もしくは思い込み)」で失敗するケースは結構多い。サターンで酷評されている「大冒険〜セントエルモスの奇跡」だってそうだ。このゲームはRPGに貿易や船のチューンナップ，はては海戦シミュレーションまで盛り込んだ意欲作だった。だが，あまりに要素が多すぎゲームバランスがメチャクチャになってしまったことに加え，マップが肥大しすぎてしまったのだ。たぶんゲーム作りに不慣れだったんだろうな。着眼点は決して悪くなかったと思うんだけどね。

鉄の時代－構造的クソゲーの登場

じゃあ，すべてのクソゲーに悪意がないのか，といえばそれは嘘。悪意とはいわないまでも，かなり姑息な思惑があって作られているものも多数存在する。それは主にタレント系の人々を題材にしたものに多く見受けられるパターンで，この人を使えばそれだけで10万本はいくだろう，ってな感じでゲーム性はスカスカ。

古くはファミコンの「光GENJI〜ローラーパニック」あたりから現在に至るまで，脈々と受け継がれているパターンだ。それにしても「光GENJI……」は内容も終わってたけど，売り方もひどかったな。なんたってCD屋にまで置いてあるんだもん。ニューアルバムと間違って買っちゃった人，結構いたんだよなぁ。

あと美少女ゲーと銘打っているものにもクソゲーが多いよね。これは「ちょっとえっちっぽくて可愛い絵があれば売れるのさ」みたいな，思いっきりよこしまな考えで作られていたりするケース。

女の目からいわせてもらうと，どういうわけか男ってのはえっちな絵を見るだけじゃなく描くことも好きだったりするからね。まさに自分の描いた絵を見せたいがためだけに作ったんじゃないか？　と勘ぐりたくなるほど，ひどいストーリーやシステムが横行してる。これも男の純情(スケベ心？)を逆手に取った確信犯だよねぇ。

そうそう，確信犯といえばこのゲームを忘れちゃならなかった。サターンの「スーパーロボット大戦F〜完結編〜」。確かにゲーム内容は文句なくイイのだが，問題がひとつ。バグるのだ！　普通ならバグごとき，と思うのだが，私は前作「スーパーロボット大戦」で突然画面が真っ暗になり，リセットもきかないバグに陥ったことがあったのだ。しかも電源を入れ直してみると，本体データが全部すっ飛ばされているではないかっっ！

そしてそのトラウマを抱えたまま「完結編」をプレイ。するとまた同じパターンでバグりやがった。幸いそのときはデータは飛ばずにその面からのやり直しで済んだのだが，どう考えても同じパターンでバグるってのはヤバくないかい？　さんざんパソ通の書き込みとかでもバグに関するものが出てたってのに，それが直されてないってのは，そりゃ確信犯だよ。もちろん私はサタマガのレビューでバグの件をちょろっと書いたんだけど，お家事情が許さないのか，編集長の目に触れることなく，その原稿は抹殺されてしまった。でも，あれでデータ飛んじゃった小学生とかいたらかわいそうだよなぁ。本来レビューってそういうことも防ぐためにあるような気がするんだけどね。ま，売れ線のゲームはいろんな意味でツラいんだってことさ。

ゲームの売り方に問題はないか

しかし売れているからって，必ずしも面白いゲームとは限らない。いや「スパロボF〜完結編」はいいゲームだよ。バグるけど(しつこいって！)。そうでなく単純に知名度だけで売れてしまうケースってこと。これって昔はあんまりなかったコトなんだけど，最近はゲームのCMが頻繁に流れるからね。人は目につくものに弱いから。深夜にハウスのカレーやらいただきレンジのCMが多いのも，見りゃ食べたくなるってわかってるからやるんであって，ゲームだって同じなわけだ。

宣伝が多いからってPSの「にゃんとワンダフル」がいいゲームだとは私にゃとうてい思えないが，買ってしまう人が相当数いるってのが事実なのだ。それもゲームに不慣れな女性ユーザーだったりね。まぁゲーム業界全体が商業主義的になってきてるから，なにより売ることが大事になってきちゃったってことだよね。内容がそこそこでもCMさえあれば力技で売っちゃうぞ，ってね。

かの「FF VII」だって買った人こそ多いけど，クリアした人って私の周りの一般ユーザーに聞く限りは半分ほどしかいなかったりする。かくいう私もゲームライターのくせにクリアしてないんだよね，実は。だってキャラが画面の奥までいくと，米粒よりもちっちゃくなってわかんなくなっちゃうし，ミニゲームも雪山(壁？)登りとか結構タルいのがあって，途中で気がそがれちゃったのよ。

同じポリゴンものでもサターンの「グランディア」にはハマれたんだけどね。あのゲームはスキルシステムを筆頭にシステム全般がかなりいいデキだったし，なによりキャラの表現の仕方がうまかった。前半中だるみ的なところはあったにせよ，キャラの個性と勢いで引っ張ってってくれたし。ま，ヤリ込んだせいかクリアに100時間近くかかったってのは尋常じゃなかったけど。

しかし結果的にCM戦略が功を奏し，いまやPSはメインコンシューマー機へと昇りつめた。思えばPSは発売当初からCMの多いマシンだったが，一昨年大量に流されたPS本体の宣伝およびソフトのCMは，ほかのマシンのそれに比べるとずいぶん洗練されてる感があった。いままでゲームに嫌悪感を抱いていた婦女子にすら「面白そう！」と思わせるCM作りのうまさ。そしてその「面白そう！」という彼女のつぶやきを聞いたゲーム好きの男は，彼女と一緒にゲームができる，もしくは彼女に喜んでもらおうと「じゃPSでも」とPSを買って家路につくのであった……。

いや，ホントにプレイしたいゲームがあってPSを買った男の子はもちろん多いよ。全体の6割はそうなんだと思う。でもこのパターンも結構アリなんだな。私の周りの新婚夫婦&半同棲カップルって8割方このパターンだもの。ソニーが狙ったのは女の子を含むライトユーザー層(およびそのお金の出所)。そしてそのもくろみは見事に大当たりし，大量のCM戦略でシェアを広げたPSはいまや軌道に乗った衛星のごとく安定した市場を広げている。

だが「面白そう！」なソフトが本当に「面白い」かどうかは別問題だし，洗練されたCMがゲームの内容を正確に伝えているかといえば，必ずしもそうとはいいきれないのも事実だ。

例えが古くてなんだけど「IQ〜インテリジェントキューブ」というゲームを覚えているだろうか？　あのゲームも発売当初，もうこれでもかってぐらいCMが流され，私の周りでも買った女の子が驚くほど多かった。しかし買ってみたはいいものの，プレイの仕方がとんとわからないときたもんだ。なんとかやり方がわかって，ああめでたしと思いきや，「つまんなくてやめちゃった」ってさ。

でもこれはなにも昔に限ったことではない。むしろ，市場がPS中心に安定しているいまこそクソゲーをつかまされる危険性があるのだ。ゲーム機がほぼPSひとつに絞られたのなら，当然ながら開発側もPSでのソフト販売に焦点を絞る。いままでサターンとPS両方のソフトを開発してきた会社なら，1機種に絞ればいい分だけ余分にPSのソフトを開発しようとするだろう。そうしてどんどんPSの販売予定タイトルが増えていく。

一見するとPSユーザーにとってもウハウハな

状況だ。でもちょっと考えてみてほしい。いま現在PSでは月に20本あまりのタイトルが発売されているのだ。夏休みや冬休みといった長期の休み前ともなれば発売タイトルはその3倍強。この膨大な数の中からいいゲーム，もしくは自分にあったゲームが選べるものだろうか？　どう考えても無理がある。専門誌とかの情報だって希薄になるし，判断基準がなくなれば派手なCMをしているゲームは売れているゲームなんだろうと思い込んでも無理はない。

しかもPSの購買層にはCMにひかれて買ったライトユーザーがごまんといるのだ。パッケージとタイトルだけで買ってしまいかねないこのユーザー層にCMでも見せようもんなら，どんなクソゲーだって（知らないのだから）買ってしまうだろう。

かくて私らゲーマーがクソゲーと思っているものですら，10万，20万本と売れてしまうのだ。しかも猫も杓子もPSで出す，つまりゲーム開発に不慣れなメーカーもPSで出しゃ売れる，とばかりにゲームを作り始めるもんだからもうお手上げ。いまはまだ売れる時期だからいいよ。でもこのままタイトル数が増え続けたら，たとえクソゲーでなくてもタイトル数が多い分だけ買う人が割れ，1タイトルにおける販売本数は確実に減っていくだろうね。

そのうえクソゲー率が上がったら，間違いなくユーザー離れが起きる。誰だって，そのときいちばん面白そうなゲームを買うものだ。テレビでCMとかバンバンやってて面白そうだと思って買ったゲームがつまらなかったら，どうなるだろう。あんなに派手に扱われているゲームですら，この程度なのかと思われはしないか？　また，バブリーな時期にこういう商売をされると，ユーザーだって学習して用心深くなる。結果として，超有名どころのゲーム以外には手を出さなくなってくるものだ。バクチで負けが続けば誰だって手堅くなる。それは当然だろう。そして，大昔ファミコン全盛期から衰退期にかけて起こったあの飽和状態ゆえに売れない悲惨な状況が再びくるのか？

これを防ぐにはハードメーカーのソニーがよほどしっかりしないといけなくなるだろう。ソニーのチェックがどの程度なのか私にはよく見えてこないのだが，メーカーの持ち込み数が増えた分だけ，許容ランクを上に設定しないと，ファミコンと同じ飽和状態に陥ってしまうだろう。ぜひぜひリーダーの重みを踏まえて頑張ってもらいたいところだ。とはいえ，ゲーム機なんて4年ももてば十分なんだけどさ。

それでも業界を目指すあなたへ

しかしゲームが生活に浸透してから10年以上も経つというのに，まだまだこの業界ではPSの「パラッパラッパー」のようにセンス一発でイケちゃう作品は少ない。そういうセンスを持った人がまだゲーム業界の中で育ってきていないのだ。

いや，ひょっとしたらゲーム業界にどっぷり浸かっている人には考えつかないものなのかもしれないな。1日12時間も開発してりゃぁ，そりゃあ枯れてもくるよ。やっぱりゲーム開発にも多少の遊びゴコロは必要なんだってことだ。

たとえばソシエッタ代官山という会社。サターンの「THE 野球拳スペシャル」や「王様げーむ」を出してる会社といえば，わかる人もいるかもしれない。どっちもえっち系のゲームなんだけど，とにかく徹底してバカくさい。ジャンケンだとかあっち向いてホイだとか勝敗も単純だけど，このくだらなさというか力の抜け加減がなんともよく，「うわ～，バカくさ～！」とかいいながらもプレイを繰り返してしまうのだ。ソシエッタの社長曰く「ゲームはアソビですから」。そこにクソゲーとバカゲーの大いなる違いが出てくるのだ。

そう，バカゲーとはたとえくだらなくても，プレイヤーがついてこれるような客観的な視野を持った人々が開発したもの。対するクソゲーは，プレイヤーの存在を無視するとはいわないまでも，開発者の世界観やゲーム感を押しつけているにすぎないゲームなのだ。開発者がどれほど高尚な理想を掲げて開発したにせよ，プレイヤーにその意図が伝わらなければ，そのゲームはプレイヤーから，ひいては世論からクソゲーの烙印を押されてもしかたのないことだろう。

ゆえに開発者はいついかなるときでも客観的視点，しかも時代のニーズにあった視点が求められるのだ。ただこれはいうのは簡単だがやるのは難しいのが現状だ。なにせゲーム会社とて会社には違いない。当然規定の出社時間はクリアしなければならないし，ゲーム開発が追い込みなら残業月200時間なんてザラに起こる。女作る時間だってありゃしない。そんな中でアソビだのゆとりの時間を確保するほうがムリというもんだ。センスだ客観的視野だという前に自分の身体の心配をするほうが先にくる。そのうえ上司がろくにゲームをプレイしたことのない人だとしたら，状況はさらにキビしいものになるだろうな。なぜなら「もっと面白いものを」というだけで，具体的な案や打開策が提示されないのだから。私はゲーム会社の上司の方々には，良作や話題作だけに限らずせめて年に50本のゲームはヤリ込んで（プレイではない。ヤリ込んで，だ）ほしいと思う。そうすることで自社のゲームの欠点も見えてくるだろうし，研究すべき課題も増えて次回作へのステップアップにもつながるはずだ。ぜひぜひ業界全体のレベルアップのためにもお願いしたい1点だ。

これから先もゲームはもっともっと増えていく。そんななかでゲームを作ろうと思うのなら，やっぱりセンスを磨くのがいちばんだと私は思うな。

ゲームセンスなんて天分のものじゃないか，と思う方もいるかもしれない。確かにそれも一理ある。天才的なアイデアの持ち主ってのはやっぱりいるからね。

でも自分がそういうタイプの人間でないと思うのなら，ゲームアイデアの面以外のところを磨けばいいのだ。グラフィックでもプログラムでもなんでもいい。それもなんとなく手応えがなかったら，なにごとも経験が大事と思ってなんでもいろいろ試してみればいいだけのこと。

たとえばこれまでくだらない，と思っていた連ドラを見るとかでもかまわない。ヒロインの心情やなぜこれが女の子にウケるのかを考察するつもりで見れば，それは立派な勉強だろう。その経験が何年後かにギャルゲーを作ったとき，ひとりよがりな思い込みや上っ面でないセリフを吐かせられるようになるはずだ。要はなんでも肥やしにしてやるという意気込みが大切だってこと。そうしたなかでいままでのゲームとは違った観点が見えてくるかもしれない。でもってそれはなにかの亜流やシリーズものだらけのゲーム業界においては非常に貴重なアイデアとなりうるかもしれないのだ。よっしゃあ，頑張れ，みんな！

謎の少女ルーシアが印象的なルナ2のオープニングアニメーション

AI機能搭載で，信頼してもらうまでがなかなか大変だけど，現実もそんなもんか？

なんかメーカーに片寄りがあるけど，サターンのRPGグランディア。これも力作だ

ゲーム自体のデキはいいんだけど……

Flipper Pinball Stories その1 フリッパーの誕生

市川幹人

洒落たカッコよさでは抜きんでていても日本では流行らないピンボール……と思いきやそうでもないらしい。メンテナンスは大変だが製品寿命が長く，ちゃんとしたスポットに設置すれば最新ゲーム並みのインカムをもたらすという隠れた人気ゲームだ。

日本ではめっきり数の減ってしまったフリッパーピンボール。実物を扱うアクション性のあるコインオペゲーム機[*1]で1ジャンルとして成り立っているものはほかにはなく，現在世界大会も行われているほど奥の深いゲームだ。

メカとアートとサウンドとゲーム性が絶妙に融合しており，ピンボールを上級プレイヤーが打つ姿はマシンと人との一体感を感じさせ，機械は人のために創られていることを改めて認識させてくれる。

フリッパー登場から50年近くの年月をかけ，ピンボールはさまざまな障害を乗り越えてゆっくりと進歩してきた。50年かけて蓄積されたアイデアは非常に多岐にわたり，たとえばビデオゲームの3機設定というのも，得点によって残機が増えたりするのも，ピンボールを真似たフィーチャだということがわかる。さまざまなアイデアが取り入れられ，または捨て去られて確固たる形に磨き抜かれたのが現在のピンボールの姿だ。ゲームについて話をしていくうえで，ピンボールについて知ることは無益ではあるまい。ここではそんなピンボールの歴史と魅力を紹介していこう。今回はピンボールの発祥からフリッパーの登場までを紹介する。

ピン（針）＆ボール（球）

19世紀初頭にヨーロッパで発生したバガテル（Bagatelle）というゲームがピンボールの先祖といわれている。ボールを打つのにビリヤードで使用されるようなキューを使用し，フィールドとプレイヤーの間にガラスも入ってなかった。針金（ピン）の立ったボード上にボールを転がし，どこに落とすかを競うゲームだ。台を見ると，揺らすのは当然という構造をしていることがわかる。

1903年（1901年という説もある）Log Cabinという最初のコインオペのバガテルがアメリカのThe CailleBrothers Companyから発表された。このときすでにフィールドはガラスに覆われ，右にあるレバーを引きボールを打ち出すという造りになっていた。以降ピンボールの歴史はほとんどアメリカにより築かれる。

日本には1920年（大正9年）にアメリカから伝わる，アメリカの25分の1ほどの国土のせいか縦形で固定され，その後，1930年（昭和5年）に愛知県警察部より許可を受け名古屋の「パチンコ店」第1号店が登場し現在に至る。

一方，ピンボールにフリッパーが登場するにはまだ時間を必要とする。

1931年Gottlieb社（以下ゴットリーブ）がBaffle Ballというピンボールマシンを発表する。このマシンは5万台を超えるピンボールマシン初の大ヒット作となる。またこの大ヒットによりピンボール産業の基礎を確立したといっていい。この時点でゴットリーブのディストリビュータにはのちのWilliams社（以下ウィリアムス）の創設者Harry Williams氏や翌年Bally社（以下バリー）を創設したRaymond Molony氏もいた。のちにウィ

表1　米国の状況

1903年	ライト兄弟が飛行機を発明
1929年10月24日	ウォール街での株式の大暴落で経済恐慌が表面化，以降世界大恐慌となり第2次世界中まで恐慌は続く
1939年4月	テレビ放送開始
1941年5月	国家非常事態宣言
1941年12月8日	日本軍が真珠湾攻撃を行う。これに対し米国は即日宣戦。第2次世界大戦に参戦となる

1. Bagatelle（Encyclopedia of pinball #1より）
ピンボールの先祖（？）バガテル。ピンボールというよりは一種のビリヤードに感じられる

2. Log Cabin（Encyclopedia of pinball #1より）
初のコインオペピンボール！　あまり面白そうに見えないのは筆者だけではないはず。フライヤー（ちらし）もモノクロだった

3. Ballywhoo（John A. Popadiuk Jr.氏宅にて）
バリーのデビュー作。ゴットリーブやウィリアムスが人名が社名なのに対し，ボール（BALL）にYをつけた社名というのが面白い

4. Baffle Ball
ゴットリーブのピンボール1作目。この作品がピンボールの礎となった

リアムスのオーナーとなりChicago Coin（以下シカゴコイン）を買収し自らStern社（以下スターン）を設立したSam Stern氏（現在のセガ・ピンボール・インク社の社長はSamの息子のGaryである）もこのゴットリーブに在籍していた。人材の面から見てもゴットリーブがピンボール産業の基礎を確立したといえる。

1932年バリーはBallywhooを発表し，75000台を販売。大恐慌の中，安価で誰にでも楽しめる娯楽が人々に受け入れられた。翌33年，バリーは初のペイアウトマシン*2，Rocketを発表。このRocketの登場後にほとんどのピンボールメーカーはペイアウトマシンを発表した。

このペイアウトマシンの発展がのちにピンボールに大きな障害となる。当時，5000を超す銀行が閉鎖され，そのため資金繰りに困った多くの企業が業務を停止した。1933年には失業率は30％を超えていたといわれている。失業しなかった者もたび重なる賃金の切り下げが行われていた。あらゆる階級の人々が日々の生活もままならないこの危機的な状況の中，働かずにしてピンボールでお金を得るプレイヤー，販売するディストリビュータ，製造するメーカーは「社会の敵」のような扱いをうける。

第2次世界大戦に参戦した1941年12月にニューヨークではピンボール禁止条例が可決され，1942年1月21日施行された。のちにロサンゼルス，大半のピンボールメーカーが存在しピンボール産業のメッカともいえるシカゴでも禁止条例が施行された。いまでもニューヨークではリプレイやマッチなどの再ゲームの権利を与えるフィーチャーは法的には禁止されている。再ゲームの権利＝クレジット≒ペイアウトというイメージがあるのかもしれない。

ペイアウトマシンはその後，大幅にゲーム性を引き上げビンゴマシンへと進化した。ビンゴマシンだけでも1979年までに125機種が発売されたが，収益効率の面からか大幅に衰退し，現在はベルギーのWIMI GAMES社のみが新作を開発，販売している。ゲームとしての奥の深さ，台自体の持つ芸術性はともかく収益効率の面だけは日本のパチンコが素晴らしかったといえる。

フリッパーの誕生

この頃，ペイアウトマシンを開発しなかったゴットリーブはギャンブルマシンではなくプレイヤーのスキルで遊ぶゲームを模索していた。

1947年，ゴットリーブのHarry Mabs氏は決定的な仕掛け「フリッパー（Flipper）」を発明。同年10月Humpty Dumptyというマシンを発表。このマシンには現在の半分ほどの長さのフリッパーが6枚も搭載されていた。現在のピンボールではフィールド中央下部に2枚あり，プレイヤーの意志で重力に逆うことが可能な仕掛けの役割が最大限に生きる位置に配置されているが，この台では左右端の上部，中部，下部に存在した。フィールドの中にボールをコントロール可能な仕掛けはこのカテゴリを以降「フリッパーピンボール」または「フリッパー」と呼ばせ，ペイアウトマシンとは明らかに違うことを認識させた。

ウィリアムスは1949年に初のフリッパーピンボールを発表したが，このときに「ボールをコントロールするフリッパーの出現により，ピンボールは偶然と引力だけのゲームではなくなった。スコアの40～80％はプレイヤーの技量により達成された」と発表している。

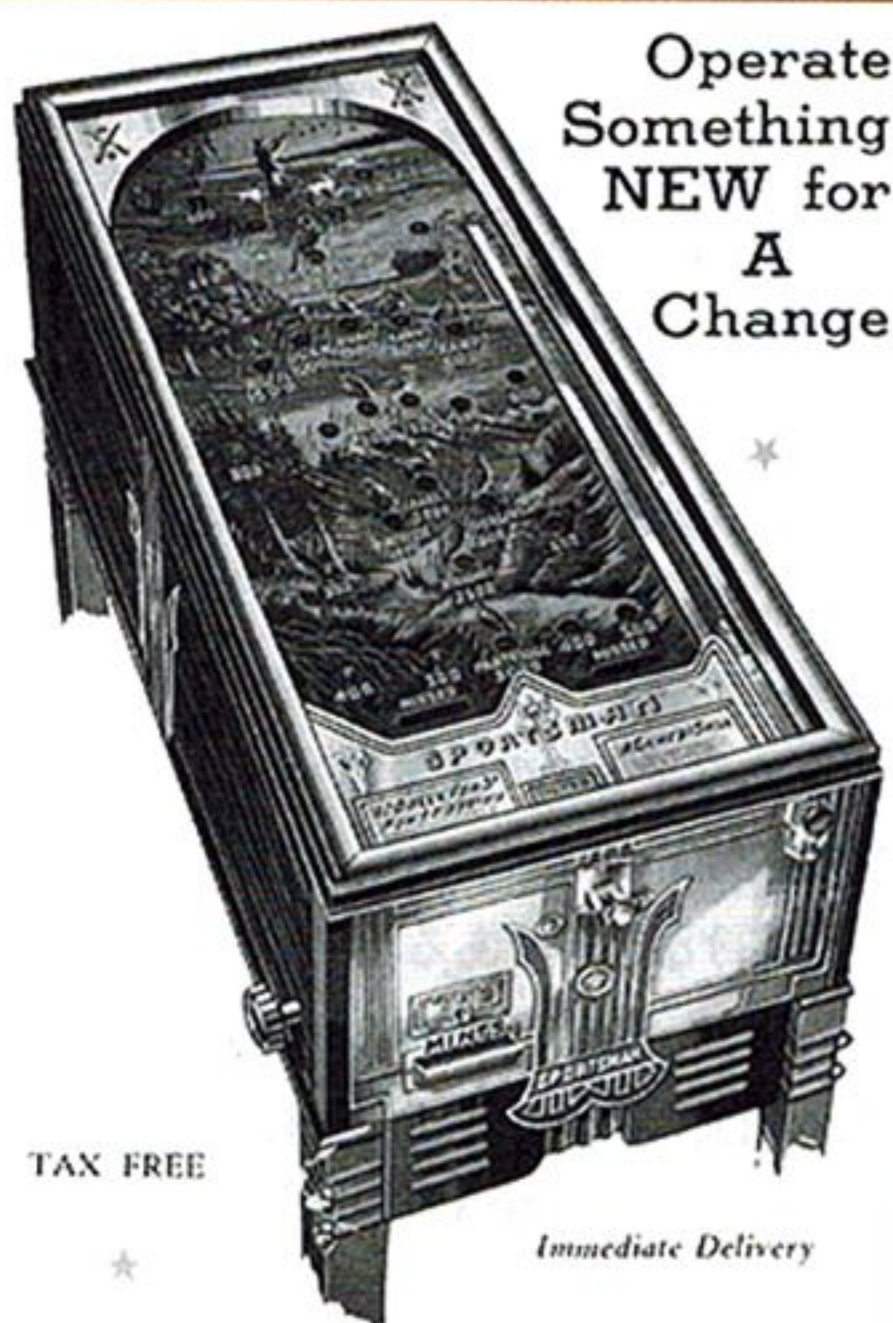
これにより，ピンボールはテクニック重視のゲームとして新しい展開をしていくことになる。

協力
TPO：出井和幸
株式会社シグマ：大久保忠雄
Williams electronics games, Planet-glo：John A. Popadiuk Jr.

＊1　コインオペゲーム機
プレイヤーがコインを投入して遊ぶゲーム機
＊2　ペイアウトマシン
得点などにより金銭の払い出しを行うゲーム機。スロットマシンもこのジャンルに属する。ギャンブルマシンともいう

column 現在プレイ可能なペイアウト式ピンボール

日本ではシグマ社の運営するファンタジアの新宿店、渋谷店、池袋店、川崎店、大宮店、川越店、草加店、志木店、水戸店にてプレイ可能。ペイアウトピンボールの完成型といえるビンゴマシンAtlantis、Artemis、Queen Of The Knight、Kassandra、Lady-X、Rebecca、Pinup Girl、Moon Light、Pastel Showerといった機種がプレイ可能。バリーのゲームを基にシグマ社の大幅なアレンジ、モディファイにより内部は当時のものではなく、CPU制御になり信頼性も抜群。最近のゲームにはない、じっくりとゲーム展開を組み立てて遊ぶテイストを一度味わってみることをおすすめする。

シグマ社のホームページ
http://www.sigma.co.jp/
Williamsのホームページ
http://www.wms.com/
WIMI GAMESのホームページ
http://www.wimi.be/

Operate Something NEW for A Change
TAX FREE
Immediate Delivery
SKILL -- SUSPENSE -- FASCINATION

5. ペイアウトマシン（Encyclopedia of pinball #2より）
'30年代当時のペイアウトマシンのフライヤー。写真でないところになんともいえない味がある

Miss Queen
EXTRA BALLS
EURAUTOMAT N.V.
Wijngaardstraat 36
B-9451 Haaltert-Kerksken Belgium
Tel. + (32) 53 83 94 94
Fax + (32) 53 83 57 46

Miss Queen
SPECIFICATIONS:
• Seven extra balls
• Five movable lines
• Quick start
• Four bonuses
• Scores till 1250
OPTIONS:
• Bill acceptor
• Card reader
• Keyswitch
• Two electronic coin acceptors.
• Protected by personal electronic key.
• Best number indication on play-field by tri colour led's.
• Data logging by Psion pocket computer.
EURAUTOMAT N.V.
Wijngaardstraat 36 - B-9451 Haaltert-Kerksken - Belgium
Tel. + (32) 53 83 94 94 - Fax + (32) 53 83 57 46

6. ビンゴマシン（Wimi Gamesのフライヤーから）
見た目は'50年代のまま発展したペイアウト式ピンボール（最新版）。内部はコンピュータ制御になり，ポケコンにさまざまなデータを転送できるところが面白い

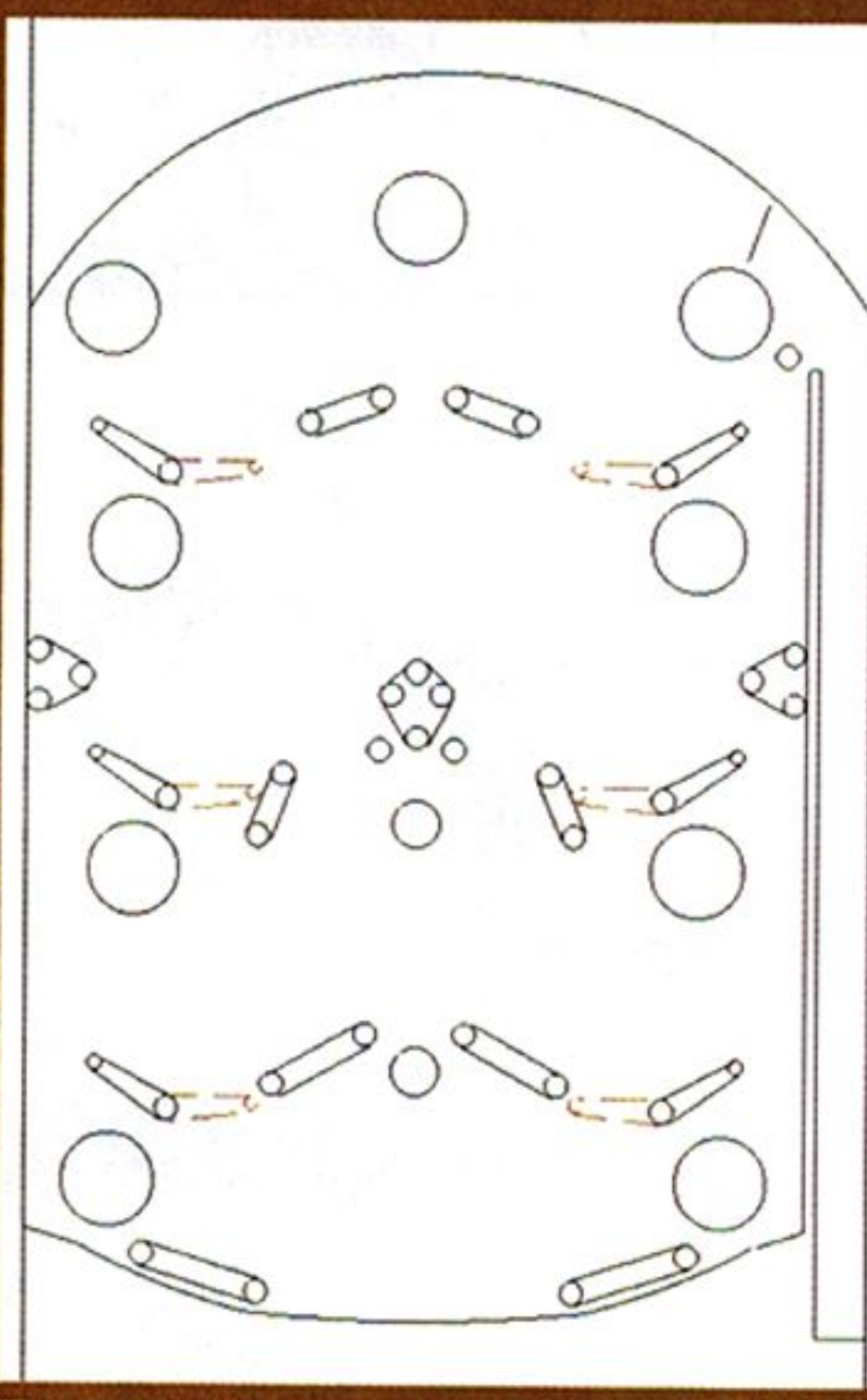



図1　Gottlieb Humpty Dumptyのフィールド図（復元版）
初搭載されたフリッパーを強調するためか，6枚もフリッパーがついている！

ドライブゲームのバランス設計

如月 緑

3D以前からジャンルとして確立されているドライブゲームは伝統的なゲーム性と3D化以降のシミュレーション性の微妙なバランスを保っている。アーケードゲーム開発にも携わった筆者による最新ゲーム分析を見てみよう。

ドライブゲームの魅力とは

車の運転そのものは，昔から遠隔地への移動手段であると同時に，より格好よく，速く走りたいといった要求を満たす嗜好品的な部分に魅力がある。実際，スポーツカーに乗る人の多くは，その運転行為そのものに魅力を感じている。

それに対してドライブゲームのひとつの魅力とは，疑似的，あるいは感覚としてリアルなドライブという疑似体験を安全，かつ手軽に楽しめるところにあるといえる。

たとえば本物の車の場合には，コーナリングをうまくこなしたときの気持ちのよい加速感は，ミスをして対向車と正面衝突や崖から転落などの危険と隣りあわせにあるものだが，ゲームではそれはない。そして，誰でも簡単に架空のレーシングドライバーになり，タイムを競ったりすることができるのである。

しかし，普通の人は，レーシングドライバーでもない限り，超高速で車を運転した経験はないはずなので，普通の人がドライブゲームを楽しむためには，その環境からの必然として，操作している車の動作・挙動が，実際に本物の自動車で走行したことのある速度の体験から延長して予想した結果の範疇に収まっている必要がある。

これは，楽しむのに最低限必要な部分であって，この部分の作り込みが甘いと，「車でない」感覚が大きくなってしまう傾向がある。

また，楽しませるということは，そのための仕掛けを作り込むということでもある。ドリフト走行をさせたいのであれば，ドリフト状態を簡単に維持できるようにしたり，ドリフト状態に入るきっかけを甘くしてみたり，グリップ走行をさせたいのであれば，直進安定性を故意に高めたりするのである。

そして，ある部分は実車以上においしい，しかし全体のうちどこかには必ず問題を含んだ「挙動」ができあがる。

その部分はそのドライブゲームを走ったときに得られるキャラクターになるのである。そして，これらは通常，そのゲームのどの部分をプレイヤーに遊んで楽しんでもらうのかという明確なコンセプトによって設計されている。

たとえばコーナーをドリフト走行させたり，対戦でぶつけあって遊ばせたり，またはリアルな挙動を楽しませることなどを企画してゲームが成り立っている。

つまり，ゲーム性としてのよさというものがあるとすれば，こういった個性が非常に明確に滲み出ていて面白い，という部分にあり，すなわちそれがそのままそのゲームの魅力であるといえよう。

業務用ドライブゲームの現状

ここ数年の間に著しく進化したリアルタイム3次元CG技術によって，ドライブゲームには非常に美しい映像が使用されるようになってきている。

かつて本格的な完全3DCGのレースゲームは，セガ社の「バーチャレーシング」でブレイクした。

このときに使用されていたCGボードは，フラットシェーディング，ノンテクスチャの四角形リンクポリゴンで秒間18万ポリゴンというものだったが，現在はハイエンドなCGボードではスペキュラー付きグローシェーディング，フルカラーテクスチャで秒間100万ポリゴン程度は通常問題なく使用可能である。

入力デバイスはそれほど進化してはいない。

ハンドルにはキックバックがつくようになったり，その制御技術は進化したものの，依然交流モーターからのステップ入力をコックドベルトを通してハンドルに反力として伝えているにすぎず，アクセルペダルやブレーキペダルもスプリングとゴム反力を使用したものである。

出力デバイスは各社とも割と工夫が凝らされている。サウンドはしっかり調整されている店舗では，4チャンネル出力によって立体的に音場が確保されているし，接触したなどのショックを振動で伝えるデバイスがセットされている。

ムービングは，CGボードのコストの影響を受けたか，どこも無難な造りになっているようだ。

しかしながら，CG技術のドラスティックな進化とは裏腹に，これらの入出力デバイスは割とおとなしい進化を続けており，映像だけでしかリアルな表現をできない状態にまでなってきている。

たとえば，ドライブゲームでリアルであること以前にいちばん大切なこととして，スピード感が挙げられるのだが，このスピード感を映像の表現だけで得るのは非常に難しいといわざるをえない。

業務用のドライブゲームは，特徴として，プレイヤーがゲームをプレイする時間に大きな制約をもって制作されている。そして，その範囲で100円なり200円なりの代価に対する満足感を常に与えなければならない。したがって，家庭用のプラットフォームであれば可能であるが業務用ではできないといった問題に悩まされることもある。

後述のGTのように，現在，家庭用のプラットフォームで本物を探求することは不可能ではなくなってきている。ハードの基本性能の飛躍的な向上に加え，入力デバイスもアナログデバイスが普及してきている。家庭用のドライブゲームでは，その制作コンセプトさえ許せば，そこそこに精巧な物理計算モデルを作ってリアルな挙動をするゲームが作成可能である。

しかし，業務用でそれを実行することは非常に難しい。業務用ゲームは，映画館のスクリーンと一緒で，その場所（＝アミューズメントパーク）に行かなければプレイできないのが普通である。したがって，わざわざ出向いてお金を払ってまでやる価値がなくてはいけないのである。リアルであることを追求すると，行き着くところは本物だ。本物をプレイしたければ，本物で首都高や峠を走ったりするほうが楽しいに違いない。リアルとは別ベクトルの価値観，ゲームとしての面白さが必須となる。

そこで，これはバーチャレーシングのころから

ずっとそうなのであるが，業務用ゲームでは，本物の挙動に近いところまで挙動計算ロジックを作り込んでから，ゲームとして楽しめるように変更と調整を繰り返すのである。

また，これは気がついている人も多いとは思うが，たとえば，スピード感を出すために，視野角を本来より広くとる補正をする場合がある。そうすると，画面の端が引き伸ばされて，物体の移動量を誇張することができるのだが，その結果，見た目の距離感覚とスピードと物体の大きさに不釣り合いが生じる。こういった補正は，スピード感を体で感じることができないこと，また，現状のポリゴンCGボードでは，ブラーの表現に多大なコストが必要であることなどがあり，なかなかなくすことができない。

家庭用ドライブゲームの現状

家庭用テレビゲームでは，次世代機と呼ばれた32ビットハードウェアの普及によって，かつての業務用CGボードに勝るとも劣らない映像を得ることができるようになった。家庭用CGゲームハードの特徴としては，その性能については，見た目にわかりやすいエフェクト機能の充実が挙げられるだろう。しかし，メモリ量やCPUパワーではいまだに制約も多く残っている。限られた範囲内でどうゲームを組み立てていくかという手腕が問われることになる。

これまで家庭用のゲーム機で入力デバイスは，弱点といわざるをえない状況であった。たとえばハンドル操作をパッドの左右キー入力で代用することは，ドライブという操作が，ある種リアルタイムのフィードバック系であることを考えると非常に感覚的に違和感を覚える原因になるものである。

しかし，最近では徐々に進化し始めており，振動のフィードバックや，アナログ入力(この場合のアナログとは，十分な分解能を持ったデジタル入力)も可能になりつつある。

出力デバイスは，家庭用である以上，特殊なデバイスを用いるわけにはいかないので，そのプラットフォームのベンダが用意でもしない限り，通常動作環境はまちまちである。この部分については家庭用ゲームで改善することはコストの問題もあってほとんど不可能ではないだろうか。

ゲームデザインとして家庭用ドライブゲームを見た場合，業務用と異なり，コンセプト次第ではいかなるジャンルのゲームでも制作可能である。実在するレースの忠実再現を主眼におくことも可能なら，架空のレース世界を構築することも，さらには業務用レースゲームの移植さえ可能だ。

売られ方も業務用ゲームとは違う。一度試して面白くなければそれまでの業務用ゲームとは違い，買ってしまったものは取っつきが悪くてもじっくりやり込んでくれる可能性が高い。半面，プレイヤーには，よさそうなゲームを購入すること以外，選択の余地がないため，広告，宣伝の影響が非常に大きいウェイトを占める。

しかし，実際に長期にわたって人気の続くソフトは，やはりゲームとしてちゃんと作り込まれ，遊べるようになっているからにほかならない。

以下では最近のドライブゲームをデザインの観点から見てみよう。

それぞれのドライブゲームについて

■デイトナ2

かつて全米で大ヒットしたタイトルの続編である。モチーフとなっているのはNASCARというアメリカではCARTと並んでメジャーなモータースポーツで，外形が普通の市販車両のものを使うストックカーレースの代表といえる。前作から4年が経っているが，その間に，セガ社のCGシステムボードはモデル2からモデル3に進化しており，今回はその改良版ともいえるモデル3ステップ2が使用されている。

コースは初級・中級・上級の3コースで，前作と同様に初級が1マイル程度のショートオーバルを8周，中級が2.5マイル程度のコースを4周，上級は前作よりも長く，5.75マイル程度のロングコースを2周である。前作では車種は基本的に1種類で，ATかMTかの違いしかなかったのだが，今回は難易度別に3車種(つまりそれぞれAT・MTが選べるので6タイプ)存在する。

このゲームのコンセプトは，前作と同様に多人数同時プレイでの楽しさを主眼に置いていると思われ，自分の運転している車の挙動については，どちらかというと初心者を優先した造りになっているようである。

コーナリング中の挙動変化については，ドリフトしている場合にもそれほど減速しないように工夫されており，また，ハンドルやアクセル操作に過敏に反応しないようになっているので，ドリフトしながらコーナーを曲がったほうが速い。これは，「このゲームはできればドリフトして遊んでください」という制作側の意図が形になったものであるといえよう。したがって，そういう意味ではグリップ走行派やリアル指向の強い方にはあまりすすめられない。

また，このゲームのドリフトは，どちらかというとパワースライドの傾向が強く，それ以外のドリフト操作はやりづらくなっており，これも好みの分かれるところであろう。ドリフトさせた場合の不安定さが車の運転のそれほど上手くない人でも予想できる範囲に収まっており，スピンしてしまった場合にも，ハーフスピンでとどまることなく，1回転するようになっている。

しかし，ぎりぎりでスピンするような場合，反対向きにスピン寸前の角度まで回転してから，そこで安定してしまうことがある。これは，ドリフトの状態をキープするためにアシストしている部分の副作用だろう。

画面全体の雰囲気や，挙動そのものには初代デイトナの面影はないのだが，通信プレイをすると，やはりデイトナである。ギヤドリと呼ばれる無理やりシフトダウンすることによるドリフト方法もそのまま通用する。

ただし，前作と異なるのは，前作では，あるギヤ位置からシフトをニュートラルに戻したときに，以前のギヤ位置のまま走行可能であるという不具合が存在し，それを利用して高速にシフトチェンジできたのだが，今回はニュートラルという状態がちゃんと間に存在しているので，急いでシフトチェンジしなくてはならないことである。

また，今回はギヤ比が離れていると，挙動変化が激しく，4速から1速にいきなりシフトチェンジしてスピンせずに走るのは，常人ではほとんど不可能である。このあたりは，前作をどうプレイしていたかによって評価の分かれるところだろう。

このゲームでは，スピード感の映像的な表現に関して，特筆すべき点がある。それは，地面のテクスチャについて，ミップマップのいちばんレゾリューションの細かい部分に，速度に応じて地面の粒子が流れるテクスチャが貼られていることである。

これによって，車が停止している場合には，粒子状の地面になっているが，速度が増すにつれて，漫画でいうところの効果線に相当するブラーがかかっているような効果が得られている。これは，画面写真などではよくわからないので，一度低い視点位置でプレイしてみることをお勧めする。

このゲームにおけるデザインは，どちらかというと，隅から隅へのモデルの作り込みに重点を置いているようである。画面全体が，静止画としてみた場合においても美しいCGとして破綻をきたさない絵を提供しており，プレイ画面を後ろで眺めているだけでも綺麗さが伝わる。

しかし，これは裏を返すと，プレイヤーの車以外に動くものが多く，派手な背景になっているので，自分の動きをはっきり感じにくいという問題がある。このゲームは，どちらかというとひとりでストイックに速いラップタイムを刻むというよりは，気のあう仲間たちと，全員で楽しく対戦することを目的としているため，この選択は間違いではないと思う。だが，もう少し画面を整理する工夫をしたほうがよいのではないだろうか。

全体的にゲームとしては非常によくできており，対戦はアツい。16人まで同時にプレイ可能であるということだが，残念ながら，日本国内には，そのような大規模なアミューズメントパークは存在しない。都市部の大型店舗で4台，郊外のテー

デイトナ2の画像。手前のテクスチャの違いに注目

ドリフトしながら遊ぶのが正しいゲームだ

マパーククラスでもせいぜい8台がいいところであろう。

なお，このゲームは，BGMをニューヨークで作成しており，ハードロック系の曲がチョイスされている。好みの分かれる部分だとは思うが，非常にデキはよい。

■セガラリー2

このゲームは，かつて日本国内およびヨーロッパでヒットした「セガラリーチャンピオンシップ」の続編である。ゲームは，WRCをモチーフにしたラリーレースゲームで，このゲームにおいてもモデル3ステップ2が使用され，前作を遥かに上回るハイクオリティな映像に仕上がっている。

コースは通常4種類で，デザートコース（グラベル（非舗装路）主体のコース）〜マウンテンコース（山岳道のターマック（舗装路）主体のコース）〜スノーコース（雪道）〜リビエラコース（夜のステージ・短いスーパーSS的なコースを2周）といった感じである。

車種は，カローラWRC，エスコートWRC，インプレッサWRC '97，プジョー306，ランエボV，ランチャストラトスの6種類で，時限式に前作のデルタとセリカも使用可能になるようである。

システムは，前作「セガラリー」と同一であり，デザイン的には，プレイヤーの目の行きそうな部分に非常に力を注いでいるのがわかる。たとえば，画面の中心位置，および中央上下，中央左右にオブジェクトの作り込みがなされ，普段あまり目の行かない画面斜め方向にはほとんど力を入れていないようである。これは，このゲームがプレイヤー本人にゲームを楽しませることに主眼を置いているので，プレイヤーが普段注視している部分に大きく性能を割く，結構割り切った方法である。

動くものが少ないので，自分のダイナミクスを効果的に背景から感じ取ることができる。ただし，後ろから見ていてもさほど面白くない絵に見えてしまう弱点がある。このあたりは評価の分かれるところであろう。

挙動的には，全体的に滑りやすく，前作を彷彿とさせるが，ドリフト走行時に進行方向が道なりの方向に若干補正をかけられているようで，かの「リッジレーサー」のように走りやすい。また，車種による程度の差こそあれ，FRの挙動に近い。

これは一般的な国産スポーツカーのレイアウトがFRだからそちらに寄せたのか，それともある種の理由があってそうしたのかは定かではないが，せっかく車種が選べるのだから選んだプレイヤーがもっと驚くような挙動変化があってしかるべきではないだろうか。

また，サイドブレーキがついているのだが，どうやらこれはアナログデバイスではないか，閾値を持ってスイッチ動作している可能性が高い。この点は疑問である。まず，業務用レースゲームがなぜこれまでクラッチペダルやサイドブレーキを敬遠してきたのかというと，これらが非常にマニアックな入力デバイスだからである。マニアックな入力デバイスは，装備しているだけで，ある意味イメージがひとり歩きする。これらを装備した場合，レースゲームファンが喜ぶだろうことは想像に難くないが，一般の（コアユーザーではない）プレイヤーが，知らず知らずのうちに難易度を高く予想してしまうことになる。つまり，人にもよるとは思うが，ライトユーザーは無意識に敬遠している。したがって，装備した以上，それを承知でプレイしようとするコアユーザーを満足させられるだけの機能を持っている必要がある。この点において，サイドブレーキのフィーリングについては改善の余地があろう。しかし，筐体が振動する部分はよくできており，これだけでもデラックス筐体でプレイする価値があるといえよう。

それから挙動全体としては，一般車両が雪道で遭遇する挙動に近い感じで，砂を噛んだ感じはしないのが残念である。また，「気持ちのいい走りができるゲーム」に仕上がっているようで，実車に近くなったかというと，むしろ前作のほうが実車に近い挙動を示していたように感じる。

ゲーム中流れる曲は，AM分室の以前からのゲームのセオリーどおり，avex系のBGMになっているが，これは正直いってデキがよくない。「セガラリー1」のBGMのほうが遥かに曲想がゲームにあっていたと感じるのは筆者だけではないであろう。

ゲーム全体の完成度は高いので，ひとりでアミューズメントパークでゲームをするときにはぜひやってみてはいかがであろうか。

■グランツーリスモ

国内，国外の有名自動車メーカーの使用許諾をとって実際の車名，メーカー名の出ているレースゲームだ。コースはオリジナルで，ゲームは手軽に楽しめるクイックアーケードモードや，レースに出場して賞金を稼ぎ，車を買ってチューニングしていくグランツーリスモモードなど，多岐にわたっている。

このゲームのプレイヤーカーの性格は非常に明解である。THE REAL DRIVING SIMULATORとタイトルにあるとおり，実車の挙動に忠実であることを目標にしている。したがって，画面から受ける印象として，本来の速度よりもスピード感は感じない。このことが，かえって新鮮である。しかし，リアルという以外ではゲームとしての制作者の意図が感じられない挙動でもある。

このゲーム，秀逸なデモ画面も含め，「格好いい」というキーワードが似合う。これは，商品を見た場合，誰でもわかる最大のウリである。また，技術的にもいろいろなアプローチがされている。

まず，家庭用のレースゲームで最初にリフレクションマッピング（環境マッピング）を使用していることが挙げられる。これは，業務用レースゲームでは，最初に行ったのはセガの「SCUD RACE」であったが，技術的には，ポリゴンハードでは，各ポリゴンの頂点にテクスチャデータのどの位置が対応するかというデータをリアルタイムに変更可能なハードウェアであれば，実装可能である。ただし，その場合には，テクスチャそのものを書き換えているのではないので，実際のところ，ムービングオブジェクトは映り込ませることができない。

この手の機能は本来ハードウェアで行えるのが望ましいのだが，現在のところ，家庭用，業務用の両方で実現しているのはコナミと日本IBMのCOBRA基板のみであり，しかも完全にサポートしているわけではないようである。このゲームのようにソフトウェアで実現する場合，その計算量は頂点数に比例して大きくなるので，CPU側の負担が大きくなる傾向がある。また，バイリニア/トライリニアのフィルタリングができないハードでは，小さい面積のテクスチャが部分的に大きく引き延ばされる場合があるので，あまり綺麗に見えない角度が存在する。それでも，このゲームでは，それが非常にインパクトになっていたので，与えた影響から考えると正解であるといえよう。

また，実車をゲーム中で使用するためには，オリジナルな車の各社の協力が必要になる。たとえば，筆者がかつて関わったレースゲームで使用許諾を得ようとした自動車メーカーの例では，レースゲームというものに対しての理解を得るのが難しく，個々のメーカーの許諾を取るのを断念した記憶があるのだが，このゲームでは，国内外の自動車メーカーの数多くと契約している。まさしく

ソニーの力，恐るべしといったところか。

このことによって，非常に販促活動がやりやすくなったことは想像に難くない。しかし，実車を使う場合，問題になる部分もある。それは，ゲーム制作において，そのモデルとなった車のイメージを壊すのが難しくなることである。これは，カーマニアなら当然と思われるかもしれないが，実際の自動車の挙動を再現したゲームを作った場合，ゲームとしての遊ばせどころに苦労するのである。シミュレータっていうものは，往々にして，つまらないのである。

いかにして遊ばせるかという部分を，このゲームでは，レース以外の部分に多く求めている。そのことを否定するつもりはないのだが，遊びがまったく感じられないのは，ディープな車マニア以外では，マイナスである。ただし，そういうシミュレータとしてのデキは非常によいので，仮想体験としての面白さを求めるのであれば，よいソフトである。

実際に，このゲームのセールスは非常に好調なようで，車マニアなど，普段ゲームをしない一般層にアピールできたことは，実車を使用することがうまくプラスに働いたよい例であろう。

しかし，このゲーム，操作性に問題があるといわざるをえない。ゲームプレイ中の操作性のことではなく，メニュー画面などのレース以外のレスポンスがしっくりこない。これは，ハードに搭載しているメモリが少ないのでやむをえないなどとはいいがたい。特に，クイックアーケードなどでの入力方法には疑問を感じる。このあたりは，やったことのない人間が何度も評価しないと，開発者は，どのような入力方法であっても，慣れてしまうので，感覚が麻痺してしまう傾向がある。まあ，本気でやり込めば，どういった入力方法であっても慣れるものなので，最終的には大きな問題になることはないと思われるのであるが。

総じて，とても完成度は高く，ドライビングシミュレータとしてよくできているので，おすすめである。

最後に

ドライブゲームは，基本的にはゲームとシミュレータの微妙なバランスの上に成り立っている「ゲーム」である。このことは，ゲームを開発する側からいわせてもらえるのであれば，非常に難しい部分であって，ゲームとシミュレータのどちらがよいというわけでもないのだが，遊んでもらう層をどのように見積もったかでも相当の違いになって表れてくる。

たとえば一般のプレイヤーに極力遊んでもらえるように考えるとドライブゲームでのシミュレータとしての性能は，必要最小限に抑えざるをえない。さらに，コースは極力周回コースを選び，覚えやすいような背景を置くこと，また，ゴールするまでの難易度を低く抑えることも必要であろう。

そして，車の挙動に関しては，「どうすれば速くコーナリングできるか」，あるいは「どういう操作をしてはいけないか」といった部分ができるだけはっきりしていることが大事になる。

このことは，操作がパターン化し，遊ばせ方の幅が狭くなるものの，上達した感覚が非常に理解しやすくなるので，業務用のドライブゲームでは一般性を上げる常套手段になっているようである。

ところが，シミュレータとしての性能を謳っているゲームに関しては，この手の誇張手段を用いると，「本物と違う」という不満のほうが前面に出てきてしまうので，当然挙動についてはできるだけ本物に近くせざるをえないのであるが，完全に同じにすると，今度はまともに走ることができないという問題が発生する。

体がGを感じていないのと，視野角などの関係で，本物と操作感覚が変わるのである。このあたりは，シミュレータ性能をいかに犠牲にせずにイメージとなんら変わらない動作ができるようにパラメータを調整できるかが勝負である。

また，業務用のドライブゲームでは，通常光ケーブルなどを使用して通信対戦可能である。通信対戦は，一般性を上げるのにもってこいなのだが，この部分のデキは対戦ゲームとしてのゲームバランスにかかっている。

普通，レースで通信対戦させる場合，レースを面白くするために，順位が後ろの人ほど，距離が離れるほど，大きくハンデキャップを与える。

実際にどうするのかというと，スピードを表示よりも速くしてあげたりするわけだが，この部分は非常に難しい。

まず，順位が後ろの人っていうのは，たいてい，一緒にゲームしているメンバーの中ではあまり運転が上手くないからその順位なのである。したがって，下手にスピードだけ上げたりすると，操作しきれなくなって壁に当たったりして余計に差を広げてしまう危険がある。

また，このアシスト量が多いと，ミスに対するペナルティの割合が下がるので，プレイヤー同士が興ざめになることもある。ここのバランスは，ゲームの習熟度がかなり異なっても，可能性として3割程度は初心者が勝てるようになっているのが普通であろう。

ドライブゲームは今後どちらに向かうのであろうか。プログラム技術としてのレースゲームの方法論は，特にCG技術について進歩することはあっても，レースゲームのルールや遊び方についてはほとんど変化していない。

これは，もうこのゲームジャンルそのものが閉塞しているというよりは，むしろ人目につきやすい一点を掘り下げ尽くした結果，ある意味での飽和状態であるといえよう。

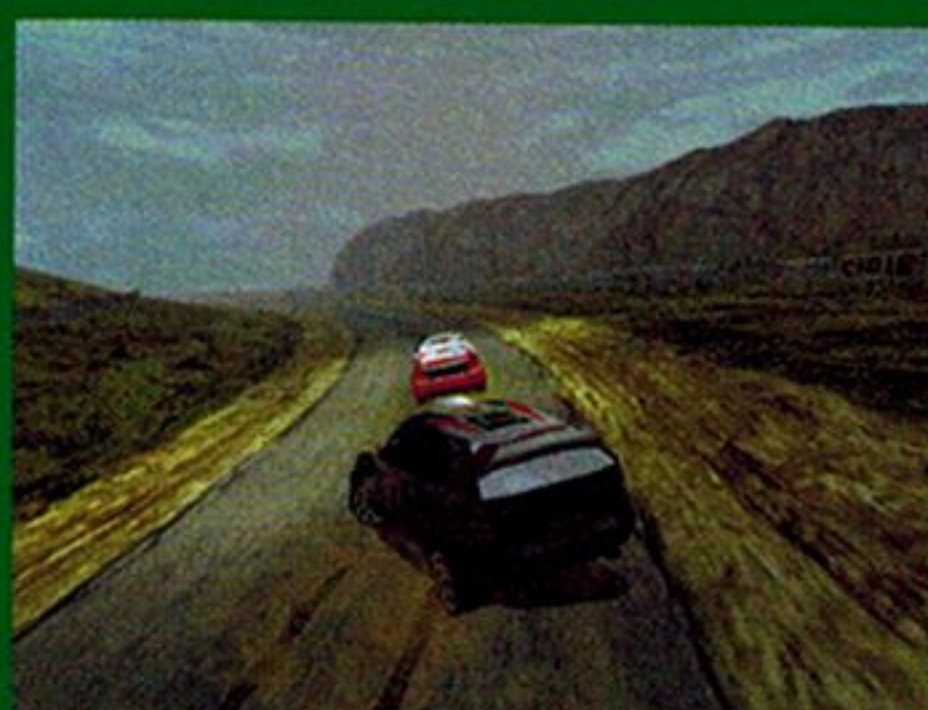
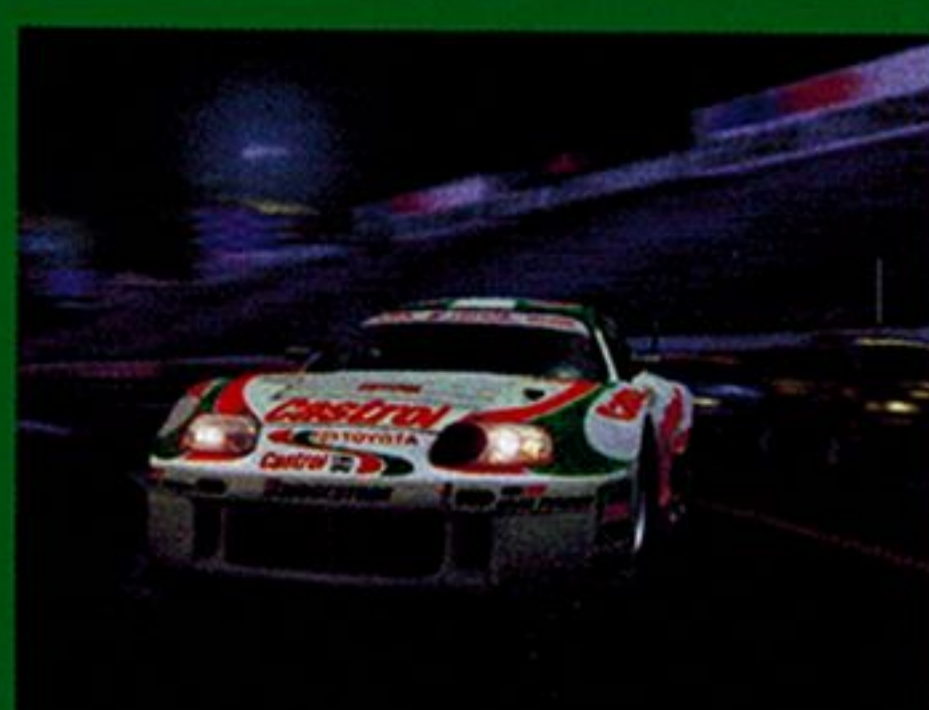
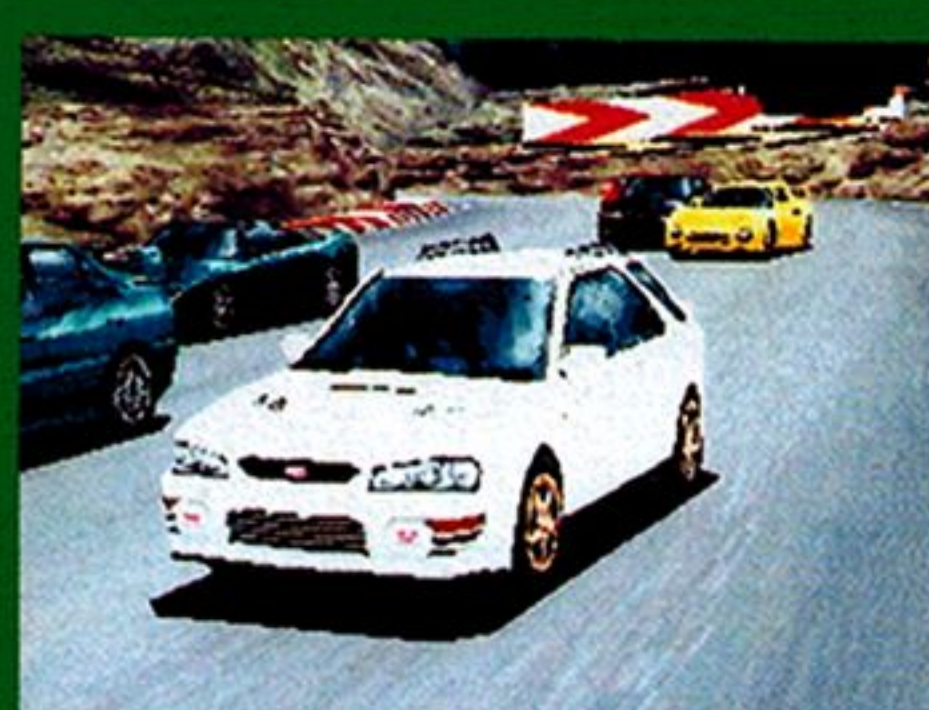
つまり，たとえば，ゲームルールとしては，通常のレースではなく，「走る楽しさ」を主眼に置いたものや，違った趣向のものが将来的に主流になる可能性もある。これは，メーカー各社の発想と努力次第である。

家庭用ゲームハードでは，その通信コストからまだ疑問が残るが，業務用ドライブゲームでは，ISDNクラスの通信回線を使用して遠隔地対戦をスタートさせることができるようになると，また新たな次元に一歩踏み出せる可能性を秘めている。

CG技術としては，そろそろポリゴンハードではないものが出てきたり（たとえば現在のリアルタイムレンダリングCGハードの弱点ともいえる光源やシェーディング方法について），もっとパワフルな性能を持つものが現れて映像としてのクオリティはますます上昇するであろう。

ソフトウェアとしてのドライブゲームは，よりシミュレータ化が進むことは想像に難くないが，やはり最終的に受け入れられるタイトルは「遊べる部分」が明確にしっかり入ったゲームであるといえるだろう。

INSERT COIN(S)
CREDIT 0
©SEGA 1998

こちらはセガラリー2「ガチャピン3」とも呼ばれているらしい……

パソコン版のセガラリー2。パソコンでもここまでの画質が出せる時代になった

GTのリプレイデータをレンダリングした，ひとつの理想形GT画像。ちなみにモデリングデータはゲーム中のものとまったく同じ。高解像度テクスチャとモーションブラーなどのエフェクトで画像はここまで変わる。やがては，これくらいリアルタイムで動くようになるのか？

背景の映り込みはまめに洗車しないと綺麗に出てこない

ワールドカップが終わっても サッカーゲームは終わらない

荻窪 圭

サッカーワールドカップの影響か，多くのゲームが発売された。しかし，まだまだサッカーの醍醐味には遠いものが多い。これからは2002年に向けたサッカーゲームが求められてくる。現在のサッカーゲームに求められているものを見てみよう。

私の友達が仲間同士で飼うことになった子猫に「ワールドカップで最後に点を取った選手の名前をつけよう」
ってことになったらしい。
「いやあ，プティでよかったね。これがカランブーとかだったら，そのネコ，一生いじめられるよ」
「ほんとほんと」

というわけでもないのだが，ワールドカップフランスでいちばん印象に残ったゴールは，ワールドカップでいちばん最後に決められたゴールなのであった。決勝戦でフランスのプティが3点目に決めたヤツだ。これは凄かった。デサイーが退場したため10人になったフランスは2点を守りきる態勢に入っていたはずだった。しかしセンターバックしてたプティが，すすーっと上がっていって，パスを受けてゴールを決めてしまったんである。チャンスと見て上がっていったプティも凄いし，そのプティを見ていてラストパスを出したフランスの選手も凄い（ビデオを見返せば誰だかわかるのだが，ビデオなんて見はじめた日にゃ，3日たっても原稿が終わらないに決まってるからやらない）。

これこそがサッカーのダイナミズムなんである。

確かに，いちばん凄かったゴールはアルゼンチン戦で“ワンダー”オーウェンが決めたシュートだ。あれはリアルタイムで体中がぞくっとした。それほど凄かったし美しかったし速かったし力強かった。ベルカンプがアルゼンチン戦で決めたシュートも身体を電流が流れるほど渋くてカッコよかった。

でも，サッカーが持ちうるダイナミズムを見せてくれたのは，驚異的な力と技と勇気を見せてくれたフォワードではなく，めまぐるしく動きながら相手守備を崩してわずかな隙をゴールに結びつけた（あるいは結びつけようとした）MFやDFたちなのである。

前編「FIFA98の素晴らしさとその限界」

だからサッカーゲームは難しい。遊ぶのが，ではなく，作るのが，だ。

野球は簡単である。プレーの瞬間瞬間に焦点を当てられるプレイヤーは基本的に「攻撃側ひとり」「守備側ひとり」だからだ。投げるときはピッチャーに，打つときはバッターに，打ったあとはボールに近い守備位置の選手とランナーに焦点を当てておけばよい。作るほうもプレイするほうもラクである。9人対9人の戦いではあるが，フィールドに同時に18人が動き回るということはないのだ。

サッカーはそうはいかない。11人対11人。双方のゴールキーパーに関しては役割が決まっているので10人対10人といってもいい。その20人がめまぐるしく動き回るうえに，ボールにアプローチしてるプレイヤーにだけ焦点を当てればいいというものではないからだ。

4-4-2やら3-5-2やら，システムはいろいろあるけれども，それは単なる基本布陣で，実際にはそんな単純なものじゃない。右サイドバックが前に上がっていけばボランチのひとりかセンターバックのひとりがカバーに入るし，フォワードがボールをもらいに下がってくれば代わりに攻撃的ミッドフィルダーのひとりが前にできたスペースに入っていくという具合。そうすることで相手のディフェンスを崩し，より有利にゲームを進めていくのである。

だから20人すべてにインテリジェントな動きが要求されるのだ。

サッカーゲームはどこまでインテリジェントになるか

プレイヤーが操作するプレイヤーは，って妙な表現だな。えっと，人間が操作するプレイヤーは基本的にひとりである。両手にゲームパッド，両足にキーボードといういでたちで一度に何人も操れる人なら話は別だが，そんな人にはなりたくない。

となると，ほかの19人をどう動かすかがプログラマの腕の見せどころなわけだ。いかにプレイしてる人間をサッカーしてる気分にさせるか。それは登場するキャラクターがリアルにモーションするってことじゃなくて，キャラクターがいかに「サッカーっぽく」動いてくれるかってことにかかっている。

この点で一日の長があるなあと感心したのが「FIFA ロード・トゥ・ワールドカップ98」だ。面

上がってきた山口がゴール前で横にドリブルしてDFを引きつけて中田にヒールパス。それを受けた中田がそのままドリブルしてゴール左隅にシュートっ。のリプレイ画面

倒なので「FIFA98」と略す。このエレクトロニックアーツのシリーズはずっと昔から，FIFA公認でFIFA96，FIFA97ときたわけだが，このFIFA98で格段によくなった。驚くほどよくなった。最初に遊んだとき，思わず「このゲーム，むちゃくちゃ凄いからレビューで取り上げましょう」と某パソコン誌にメール打ったくらいだ。

なにしろ，スピード感といいダイナミズムといい，むちゃくちゃサッカーっぽいのだ。いまでも飽きずに2日に1回は気分転換にプレーしてるほどである（CD革命ヴァーチャルなぞという怪しげなソフトを使って，CD-ROMをまるごとHDDに入れちまってるから，いつでもすぐ立ち上がるのだ）。それほど気持ちよく遊べる。

面白くなった最大の理由が，ボールを持っている（＝人間が操作している）以外のプレイヤーの動きである。わかりやすく日本代表でいうと，城がボールをもらって左サイドへドリブルして開いていったとしよう（ゴールに向かう道は相手のセンターバックにマークされているから，サイドへと逃げていくわけだ）。すると相手の右サイドバックやセンターバックはそれに釣られて城を追いかけてくるため，中央にスペースが空く。城が左サイドに開いてくると左サイドにいる名波は中央のスペースへと寄っていくので，そこへ城からパスを出すと名波はフリーの状態でゴールを向けるというわけだ。で，シュートである。

ああ，ちょっと前までのサッカーゲームなら，ここで城と名波のポジションが重なっていたものなのに。

同じように呂比須がゴール前でボールを受け取って横にドリブルする。すると相手ディフェンダはそれに釣られてゴール前にスペースを空けてくれるので，そこに中田が走り込んでシュートする。

ああ，ちょっと前のサッカーゲームならスペースに走り込んでくれるなんてこざかしい真似は一切してくれなかったのに。

ゴール前に相手ディフェンダが密集しているのでボールを後ろに戻したら，そこに井原が上がってきていて，ロングシュートを打つ，ということすら可能なのである。

FIFA98では選手がこういう「サッカーらしい」動きをしてくれるのだ。それがうまく決まったときの気持ちいいことといったらっ。

これは選手の動きがよりインテリジェントになった，ことを意味してる。FIFA98の場合はゴールキーパーの動きがあまりにダサダサで思わぬタコな失点をしてしまったりするというツメの甘さを残しているのだが，それはFIFA99で直されているだろう。

後編「インテリジェント化による弊害とそれを打破する可能性」

動きがインテリジェントになる，というのは，人間が操作する以外のプレイヤーだけに限ったことじゃない。人間が操作するプレイヤーにもそれはいえるのだ。昔のサッカーゲームはパスを出す方向や強さも人間が決めてやらねばならなかった。ちょっと考えれば無謀ってことはわかる。ゲームパッドは「8方向」しか持ってないのだ。実際の試合でそんな都合のいい方向に味方がいてくれるわけがないし，そんなに都合のいい方向にゴールがあるわけでもない。

そこでプログラマは「十字パッドの押された方向に近いところにいる味方に向けてパスを出す」というプログラムを組むようになる。この段階で多少思いどおりにゲームを組み立てることが可能になってくる。問題はどのくらい余裕を持たせるか，だ。けっこう明後日の方向を向いていてもちゃんとパスとしてボールが届くようになると，残念ながらゲームとしての面白みは半減する。足下へのパスを繰り返すだけのサッカーはダイナミズムに欠けるからだ。

PlayStation用の「実況ウイニングイレブン」シリーズはそれをやりすぎたために，スルーパスボタンを設けるに至った。要するに，足下へ合わせるのではなく，味方の前にあるスペースを狙おうというわけだ。遊ぶ側としては「スルーパスな気分」が味わえるだけで，実際に試合が高度になるわけではない。

あまりに操作するプレイヤーの動きがインテリジェントになると，結局人間はある程度の方向づけをしているだけで，ほとんどをプログラムがこなしているという妙な状態になり，「試合に参加している悦び」がどんどん失われていくのだ。それは本意ではない。

それに比べるとFIFA98はかなり絶妙なバランスを保っており，それに加えて選手の能力に応じた「ブレ」がよい味を出しているが，インテリジェントさが運んでくるジレンマから完全に逃れることはできない。

ついでだから比較しておくが，1つひとつの選手の動きでは「実況ウイニングイレブン」のほうがリアルだ。でもそれは結局のところ，どうでもよいことなのである。リプレイ時はリアルな動きをしてもらったほうが楽しいが，実際のプレイではそれよりもフィールド全体を見渡せることや，選手のポジション取りがよりサッカーらしいダイナミズムにあふれていることのほうがずっと大事なのだから。

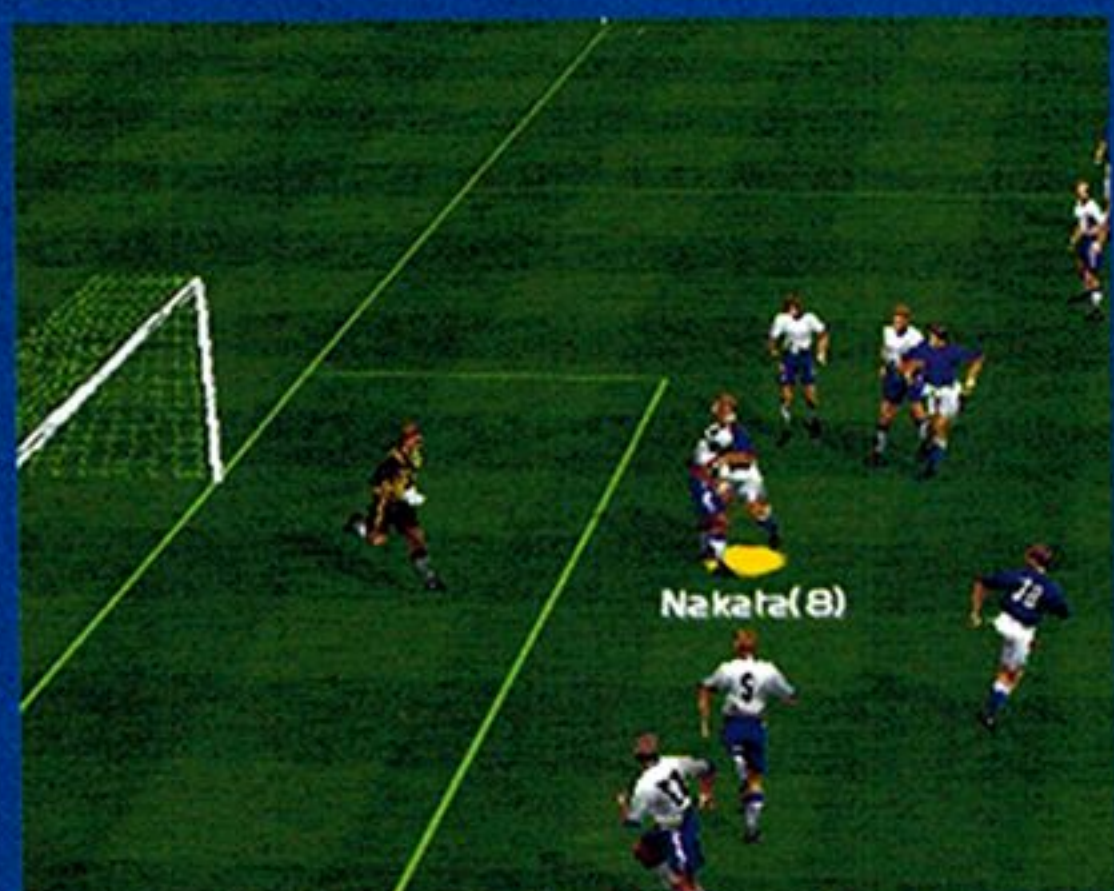
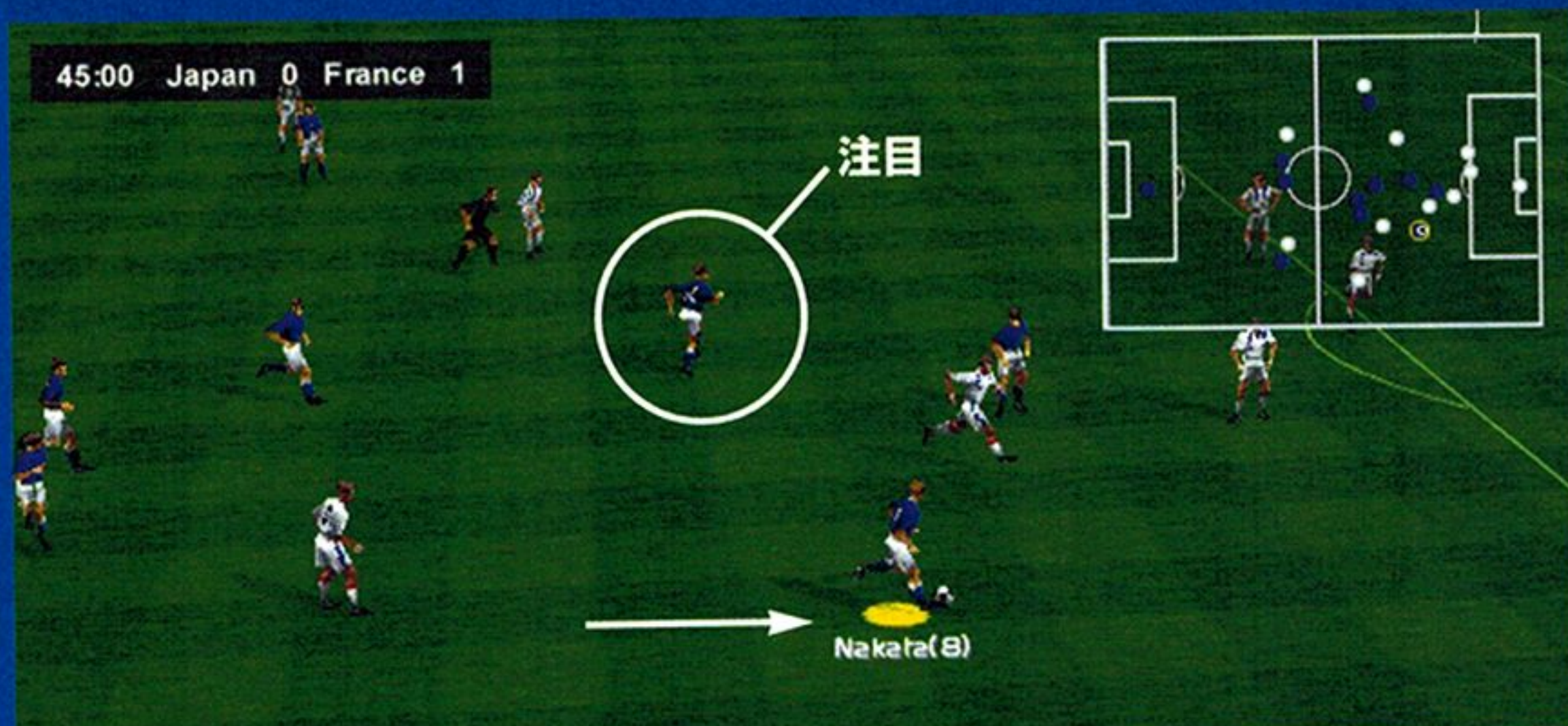
ひとつの目安として，「フォワード以外の選手がどれだけ点を取れるか」というのを挙げてもいいだろう。ほとんどフォワードしか点に絡めないってことは，ほかの選手がゴール前へ飛び込んでこないということであり，それだけ古くさい型にハマった動きのアルゴリズムしか持ってないってことだからだ。

根本的な問題を解決するには

インテリジェントになりすぎる弊害は，基本的にボールを使った団体競技ゲームすべての限界だ

45:00 Japan 0 France 1

注目

Nakata(8)

ゴール前の混戦でパスを受けた中田……このあと飛び出してきたGKにクリアされる

右サイドでフリーでパスを受け，ドリブルで上がる中田。左にフリーになろうと開いていく選手がいるのに注目。ここであそこにうまくパスが出せれば大きなチャンスになる……。ならなかったけど(画面キャプチャしてる間にボールを奪われたから)。右上のレーダーに注目。4-4-2といいながら，状況に応じて選手はバラけたり固まったりする。中盤の混戦からパスが中田に出たという状況なので選手が中盤に固まっている

といっていい。それは，攻撃時はボールを持っている選手が，守備時はボールに近いところにいる選手が自動的にアクティブになるというシステムが原因である。

実際のゲームではボールはめまぐるしく動き，どの選手も次を読んだ動きを要求される。操作する選手がコロコロ変わるようではそれについていくのは難しいから，選手の動きのアルゴリズムが改良され，どんどんインテリジェントな動きが実現されていく。それによって，人間側は前述したような「ゲームに参加はできるけれども結局プログラマの手のひらで遊ばされているような感覚」に襲われてしまう。しかし，人間側にあまり多くの役割を課すと，ゲームの難易度が上がるばかりで「サッカーしてる気分を味わう」という側面が失われる。

たとえば，サッカーではボールを持たない選手のフリーランニングが重要だといわれる。でもボールを持っている選手を人間がコントロールする方式だと，フリーランニングはコンピュータがやってくれるか否かのみにかかってきてしまう。それでは，人間側の楽しみがない。なかには「小手先の技」を導入して，特殊なボタンを押すことでオーバーラップやスルーパスを使うというゲームもあるが，それは本質的な問題ではない。ただ流行の戦術を例外的に取り入れただけで根本的な解決にはなってないからだ。

では，よりゲームに深く参加できる方法はないのか。「ある」。わたしの頭の中にはある。

チーム全体を担当する必要はないのだ

かつて(もう10年も前だが)「熱血硬派くにおくん」と呼ばれるゲームのシリーズがあった。基本的にデタラメなノリだけのイケイケなゲームなのだが，そのひとつ「サッカー編」が忘れられないのだ。このサッカーゲームはほかのサッカーゲームとはひと味違っていた。人間は「キャラクターのひとりだけ」を終始担当するのだ。

これを現代のサッカーゲームに応用したらいいと思うのである。いままでのサッカーゲームでは，人間は「監督とボールに絡んでいる選手」を担当するという前提でことが運んでいた。それは考えてみると妙な話である。1つひとつのプレイが独立している野球ならともかく，サッカーのようなダイナミズムを持ったゲームだと，操るべき選手がコロコロ変わってしまうので感情移入しづらいのだ。

そこで私が提案するのは，人間は「監督とたったひとりの自分自身を投影する選手を全面的に担当する」システムである。

これは面白いはずだ。自分が操作してない選手はまったく操作できないのである。操作ボタンには「グラウンダーのパス」「ロングパス」「シュート」に加えて「パスを要求する」ボタンをつけよう。攻撃的MFの選手を選んだら，彼は中盤を走り回って，パスをもらい，パスを出し，という作業ができる。周りの選手がどう動くかは，これまでのサッカーゲームがそうであったように，設定画面で方針を決めればよい。FIFA98並みのアルゴリズムを持ってすれば，かなりリアルな動きをしてくれるだろう。

もし左サイドバックを選んだら，普段は左サイドで守備をして，チャンスと見たら，ダダーーッとオーバーラップして，適当なところで「パスを要求」してパスを受け，そのまま左サイドをドリブルで上がってセンタリングという技ができる。そのあとシュートまでいけるかどうかは，ほかの選手次第ってわけだ。

サッカーフリークならリベロをやりたがるかもしれない。チャンスを見つけて後ろから上がっていく楽しさを味わえるからだ。

チーム以上に，特定の選手へ感情移入していくというのがポイントだ。これは，ボールを持っていない選手の動きも非常に重要であるというサッカーだからこそゲームとして成り立つシステムで，もしこれを野球ゲームでやったら退屈でしょうがない。1試合に4，5回しかバッティングできなくなるのだから。でもサッカーならボールを持ってなくても徹頭徹尾(レッドカードでももらわない限り)試合に参加できるのだ。

当然，コンピュータのコントロールするほかの選手が思いどおりに動いてくれないというイライラはたまるだろう。ドゥンガやストイコビッチのイライラまで味わえるおまけつき，ってわけだ。ある程度はゲームを止めて指示できるようにしておくのも面白い(学習機能付き！)。これには「ドゥンガ機能」という名前をつけよう。でも，それもまたサッカーのうちであるし，この方式ならほかの選手の動きがどれだけインテリジェントになっても操作する人間はそれに辟易しなくてすむ。少なくともフィールドにいるプレイヤーのひとりだけは，自分の支配下にいるからだ。

この方式のメリットはほかにもある。

たとえば，スキル。ゲームに慣れないうちは，難易度を下げる代わりに，難易度が低いポジションを担当すればいい。中央の選手より，サイドの選手のほうが動きは単純であり，対応はしやすいだろう。

たとえば，多人数参加モードへの対応。参加する人間が「自分はどの選手を担当するか」決めてしまえばいいからだ。

仲間を11人集めて1チームを作り，ゲームパッドを11個用意してコンピュータ相手に戦うのもまた楽しかろう。さすれば，実践さながらのバトルが味わえるに違いない。うーん。サッカーファンなら一度はやってみたいでしょ。

さあ，誰か作っておくれ。そういうリアルにサッカーな気分を味わえるゲームをっ。

その日までわたしは「海の向こうではEURO 2000の予選が始まったなあ」などと思いながらMSのゲームパッドを握り，PentiumⅡマシンでFIFA98を遊び続けるのだよ。

特徴ベクトルのクラスタリングを利用したグラフィック処理

新美 嵐/Run Nearme

パターン認識とか特徴ベクトルだとかいった話題はパソコンとは縁遠いものがある。こういった人工知能的雰囲気の方法論は一般にはあまり活用されていないかのように思われる。しかし，実はそこから導かれるものが動画圧縮などで見たようなフォーマットだったり，デジカメやビデオチップで使われていたりすることに気づくだろう。「特徴」を探し出すことで画像などの情報量を効率的に縮小する際の指針を示してくれるのだ。JPEG，MPEGといったものも基本をたどればこういった考え方を理解するうえでも有用だ。

近年，パターン認識を利用した製品が多数リリースされるようになりました。身近なところではザウルスに代表されるPDAの手書き文字認識。筆者のへろへろな字でも実用的な時間できちんと認識してくれます。また音声認識でもPC向けのソフトが発売されています。こちらは使用したことがないのですが，店頭でお姉さんたちがやってたデモを見る限りでは，使うほうで少し手を入れればなんとか使えそうな印象を受けました。

パターン認識を少し具体的にいうと「入力されたパターンからそれが属するカテゴリを出力すること」となります。詳しい説明はこのあとやりますが，要は，「バナナを見て『バナナ』，りんごを見て『りんご』ということ」がパターン認識です。これは入力パターンをカテゴリに分類するという見地からパターン分類とも呼ばれます。またパターンに付随する知識なども機械に扱わせ，より高度な処理を行わせることはパターン理解と呼ばれます。このジャンルの話はまとめてパターン認識・理解と呼ぶことが多いです。

パターン認識は人間ならなにげなく行えてしまうことでも，機械にやらせると大変なことのひとつです。しかし，本当に大変なのは認識率を向上させることで[*1]，その要となる部分はそんなに難しくありません。本稿ではパターン認識の手法を使って，グラフィックをいじり回して遊んでみようと思います。グラフィック処理は見ためにすぐ結果がわかりますし，思いもかけない処理結果が出てもそれはそれで非常に楽しいですから。

サンプルプログラムはC++で作成しました。アルゴリズムの肝はクラステンプレートで記述し，それを各事例で利用しています。また，動作確認はFreeBSD 2.2.2R + gcc 2.7.2.1で行っています。IRIX 6.3 + CC（SGIのC++コンパイラ）でも問題なく動きました。特定環境に強く依存したコードではないので，ほかのUNIXはもちろん，通常のC++環境があれば動作は容易だと思います。

＊1　たとえば「認識率95％」の文字認識器があったとします。一見これはよい数字のようですが，実際には「20文字に1文字の率で間違える」ということになります。どんな文章になるか想像がつきますね。ここから1％でも向上させることが認識屋の腕の見せどころです

■本の判型を分類するのだ

グラフィック処理に入る前に，身近なものを対象にしてパターン認識の流れをつかんでみましょう。つらつら考えて，部屋に転がっていた本の判型（サイズ）をコンピュータに分類させてみることに決定。さっそく判型の異なる本を適当に何冊か集めてみました（写真1）。

本の判型はA4からA6に分類できそうです。変形判を除けば縦横の長さのうち片方の情報さえあれば判型は決定できます。よって縦の長さのみで分類してみることにします。

各判型の縦の長さは表1のとおりです。

そして実測した本の縦の長さは表2のとおりでした。便宜上イロハで識別子をつけます。

これらイからトまでの本がA4からA6のどの判型に当たるのか，分類するにはどうすればよいでしょう？　……ちょっとわざとらしすぎますか。本の縦の長さと各判型の縦の長さを比較し，いちばん近い判型を選べばOKですね。このような比較による分類は，パターン認識のポピュラーな手法です。

以上の話を代表的なパターン認識処理の流れに当てはめてみます。高校生くらいまでの方にはちょっとハードな説明かもしれませんが，図1を見ながら頑張ってついてきてください。

まず認識対象（入力パターン）を観測してその特徴を表す量を導出します。本の特徴を表すパラメータには縦横の長さ，質量，ページ数などいろいろあります。これらのパラメータを書き並べたものをパターンベクトルといいます。

パターンベクトルそのままでは処理しにくいため，分類したいカテゴリの特性に応じてパターンベクトルを変換し，より本質を表したパラメータ

写真1　転がってた本。いずれ劣らぬ名著ばかり

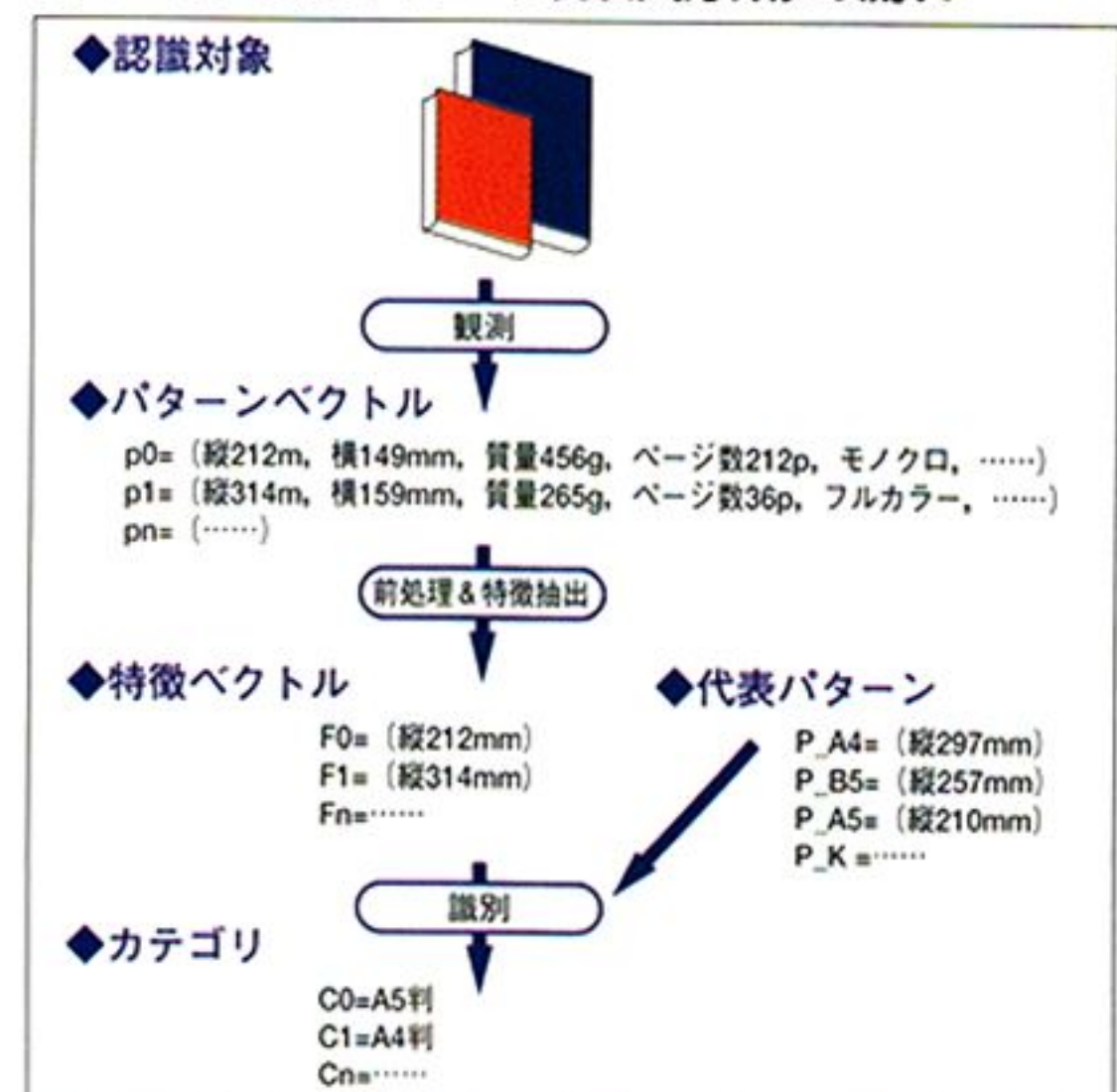
表1　判型の縦の長さ

判型	縦の長さ
A4	297mm
B5	257mm
A5	210mm
B6	182mm
A6	148mm

表2　実測した本の縦の長さ

本（ジャンル）	縦の長さ
イ（ゲーム攻略本）	297mm
ロ（Oh！MZバックナンバー）	277mm
ハ（少女マンガ雑誌）	256mm
ニ（JavaScript解説本）	232mm
ホ（C++解説本）	209mm
ヘ（マンガ単行本）	176mm
ト（SF小説）	148mm

図1　よくあるパターン認識（分類）の流れ

◆認識対象
観測
◆パターンベクトル
p0=（縦212m，横149mm，質量456g，ページ数212p，モノクロ，……）
p1=（縦314m，横159mm，質量265g，ページ数36p，フルカラー，……）
pn=（……）
前処理＆特徴抽出
◆特徴ベクトル
F0=（縦212mm）
F1=（縦314mm）
Fn=……
◆代表パターン
P_A4=（縦297mm）
P_B5=（縦257mm）
P_A5=（縦210mm）
P_K =……
識別
◆カテゴリ
C0=A5判
C1=A4判
Cn=……

の並びを抽出します。この作業を特徴抽出，抽出されたベクトルを特徴ベクトルといいます（通常は特徴抽出の前にノイズ除去などの前処理が入ります）。本の判型では縦の長さのみに絞ったことが特徴抽出に当たり，縦の長さが特徴ベクトルに当たります。よってこの例では，特徴ベクトルは1次元のベクトルになりますね。

判型に当たるものはクラスやカテゴリと呼ばれます。C＋＋のクラスと混同するのを防ぐため，以降カテゴリと呼びます。各カテゴリの標準的な特徴ベクトルはテンプレートや代表パターンと呼ばれます。こちらもC++テンプレートとの混同を避けるため代表パターン（もしくは代表点）と呼びます。

最終的なカテゴリの決定は，各代表パターンと入力パターンとの特徴ベクトルの距離を測り，いちばん近い代表パターンのカテゴリを選択することで行われます。このようなカテゴリ決定に用いる関数を識別関数といいます。識別関数には有用なものがいくつも提案されていますが，代表パターンからの距離を測る関数は，感覚的にも馴染みやすいものです（図2）。先ほど，「本の縦の長さと各判型の縦の長さを比較し，いちばん近い判型を選ぶ」と書きましたが，長さの差の絶対値が代表パターンとの距離になりますね。この距離がいちばん小さかった判型が属するカテゴリとなります。ここでめでたくコンピュータに対象の認識を行わせることができました。

さて，パターン認識の流れを踏まえて作成したC++のクラステンプレート（いきなり出た）がcluster.hです。またクラスタという新しい言葉が出ていますが，とりあえずカテゴリと同じものだと思ってもらって構いません。クラステンプレートClusterは代表点の配列proto_tableを持ち，メンバー関数discriminate()で特徴ベクトルを識別します。

これを利用して本の判型を分類するプログラムがbook.ccです。特徴ベクトルの要素は本の高さのみで，識別用の距離関数は単に差の絶対値をとったものになっています。

なにはともあれ実行してみましょう。

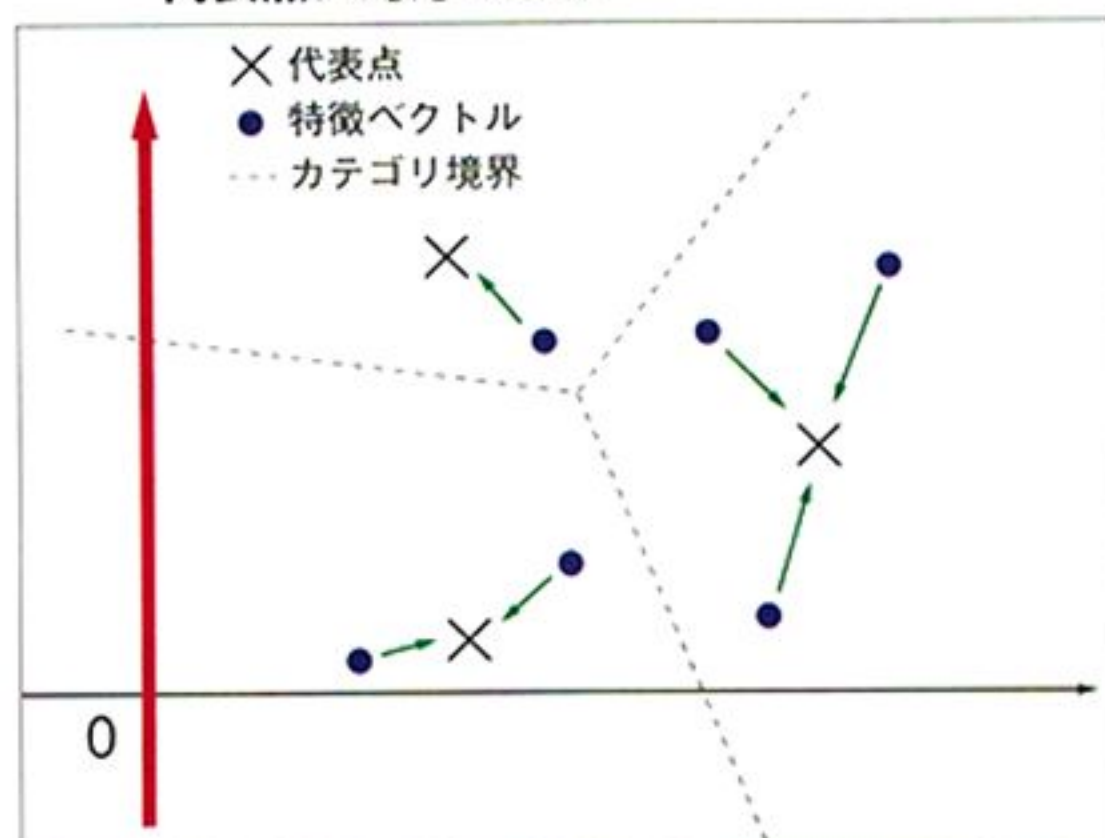
```
gcc book.cc -lm
```

でコンパイルできます。mathライブラリはなくても通るかもしれませんが，このあとのソースのこともあるので一応つけておいてください。

実行結果です。

```
real size: 297.0 quantized size: 297.0 category: 0
real size: 277.0 quantized size: 297.0 category: 0
// ロ
```

図2　特徴ベクトルの識別。距離がいちばん近い代表点に対応づける

× 代表点
● 特徴ベクトル
カテゴリ境界
0

リスト1　cluster.h

```
/*
   cluster.h
   特徴ベクトルのクラスタリングや識別を行うクラステンプレート。
*/

template<class T>
class Cluster {
  T          *proto_table;     // 各クラスタの代表点(prototypes)の配列
  int        k;                // クラスタの数
public:
  int        count_feature(T *feature_array);// 特徴ベクトルの数を数える
  Cluster(T *prototype_table) {                         // 事前に代表点をもらうコンストラクタ
    proto_table = prototype_table;
    k = count_feature(proto_table);            // クラスタ数をカウント
  }
  void  discriminate(T *feature_vector, T *quantized_vector, int *category);
                                               // 特徴ベクトルを識別する
};

// 特徴ベクトルの数を数える
template<class T>
int Cluster<T>::count_feature(T *fa)
{
  int        i = 0;
  while (!fa[i].is_end()) i++;
  return i;
}

//  特徴ベクトルを識別する
template<class T>
void Cluster<T>::discriminate(T *fv, T *qv, int *ct)
{
  int         cate,j;
  double dist, min;

  while (!fv->is_end()) {       // 特徴ベクトルが尽きるまでループ
    cate = 0;
    min = T::start_dist;        // この値から出発
    for (j=0; j<k; j++) {       // k個の中から距離最小のクラスタを選ぶ
      dist = get_dist(*fv,proto_table[j]);  // 距離を得る
      if (min > dist) {         // 今までの中で一番距離が近いか？
             min = dist;        // 最短距離と
             cate = j;          // カテゴリを更新
      }
      if (min < T::break_dist) break;        // 充分近いので打ちきり
    }
    if (qv != NULL) *qv++ = proto_table[cate];
    if (ct != NULL) *ct++ = cate;
    fv++;
  }
  if (qv != NULL) *qv = *fv;                 // end
}
```

リスト2　book.cc

```
/*
   book.cc
   本の判型を識別するプログラム。
*/

#include <stdio.h>
#include <math.h>
#include "cluster.h"

//======== 本の判型 ========//
const double EOD = -1;                             // 特徴ベクトル終端記号

struct BookSize {
  double height;                                   // 本の高さ

  static double start_dist;    // 開始距離
  static double break_dist;    // 打ちきり距離
  inline int is_end(void) {
    if (height == EOD) return -1;
    return 0;
  }
};

double BookSize::start_dist = 99999.9;
double BookSize::break_dist = 1.0;

//-- 距離関数 --//
double get_dist(const BookSize& b0, const BookSize& b1)
{
  return fabs(b0.height - b1.height);
}

//-- 各カテゴリの代表点 --//
BookSize proto_hi[] = {
  { 297 },                     // A4
  { 257 },                     // B5
  { 210 },                     // A5
  { 182 },                     // B6
  { 148 },                     // A6
  { EOD },
};

//-- 実測した本の特徴ベクトル --//
BookSize real_hi[] = {
  { 297 },                     // イ
  { 277 },                     // ロ
  { 256 },                     // ハ
  { 232 },                     // ニ
  { 209 },                     // ホ
  { 176 },                     // ヘ
  { 148 },                     // ト
  { EOD },
};

int main(int argc, char *argv[])
{
  Cluster<BookSize>  book_clst(proto_hi);
// 本の高さを扱うクラスタリングオブジェクト
  BookSize *quant_hi;                                             // 量子化された本の高さ
  int         *size_cate;                                         // 判型のカテゴリ
  int         num_book;

  num_book = book_clst.count_feature(real_hi);            // 本の数をカウント
  quant_hi = new BookSize [num_book];                     // 結果の領域確保
  size_cate = new int [num_book];                         // 同上
  book_clst.discriminate(real_hi, quant_hi, size_cate);   // カテゴリ識別

  for (int i=0; i<num_book; i++) {                        // 結果表示
    printf( "real size: %5.1f quantized size: %5.1f category: %d¥n",
            real_hi[i].height, quant_hi[i].height, size_cate[i]);
  }
  delete[] quant_hi;
  delete[] size_cate;
}
```

```
real size: 256.0 quantized size: 257.0 category: 1
real size: 232.0 quantized size: 210.0 category: 2
// ニ
real size: 209.0 quantized size: 210.0 category: 2
real size: 176.0 quantized size: 182.0 category: 3
real size: 148.0 quantized size: 148.0 category: 4
```

カテゴリが番号出力なので少しわかりにくいですが，ちゃんと距離がいちばん近い判型に分類されているのがわかります。このうちロとニの本は変形判です（ロはA4変形，ニはB5変形）。これがどのサイズの変形判かまで認識するには，ちゃんと横の長さも特徴ベクトルに含めないとだめでしょう。あるいは既知のカテゴリと距離が離れすぎるということで，リジェクト（識別不能）してしまうのもよい方法です。

以上のような対象の分類は，もちろん本来のパターン認識としてコンピュータへの強力な入力手段となります。が，もうひとつ，情報量の大幅削減という面があります。多くの情報量を必要としたパターンベクトルを，カテゴリという少ない情報量にまで落としているのですから。逆に分類されたカテゴリと代表点を記録しておくと，少なくとも特徴ベクトルまでは元の情報（に近いもの）を復元できることになります。以降のグラフィック処理の話ではこれを利用して情報の削減と復元を扱っていきます。

■マニュアルで減色処理するのだ

本の話題はこの辺にしておいて，本題のグラフィック処理に入りましょう。

コンピュータで扱うグラフィックは，周知のとおり輝度を表す画素（ピクセル）の集まりでできています。カラーなら1ピクセル当たり赤緑青の3種類の輝度値を持つベクトルで表現されます。このベクトルの値次第で黄色だとか紫だとかの色に認識されるわけです。

そろそろ聞き慣れた言葉が出てきたところで，まずはピクセルの色を認識対象にしましょう。RGBの値が特徴ベクトル，緑・青・灰色・黒・水色・ピンク・オレンジ・黄色・草色・白・茶色・赤[*2]などの色がカテゴリになります。

このように当てはめると減色処理が実現できます。これは字のとおり，多くの色を使っている画像をより少ない色数で表現する処理のことです。特殊効果のほか，画像圧縮にも使われます。X Window Systemでよく使われるグラフィックビュアのxvであれば，%xv -8 -ncols nとオプション指定するとn色に減色してイメージを表示してくれます（ディザつきです）。Photoshopならモードをインデックスカラーに変換するときに任意の色数に減らせます。そのほかのビュアやペイントツールにも似たような機能はあると思います。手近なツールでまず減色処理がどのようなものか試してみてください。

さて，これを実現する方法を考えましょう。

分類部分は先ほどの本の判型で使ったcluster.h をそのまま使います。特徴ベクトルの構造体は画像に適したものに書き換えます。それらをimage.h と image.ccにまとめました。特徴ベクトルは本の縦の長さから，RGB（赤緑青）の値を持つ3次元のベクトルに変更です。各要素は0.0～1.0の値をとります。距離関数も3次元のユークリッド距離に変えました。何次元の特徴ベクトルになっても，識別器には代表点からの距離さえわかれば特徴ベクトルの識別ができます[*3]。これが距離による識別のよいところです。

代表点は本の判型と同じく，あらかじめ分類すべき色をテーブルで与えています。本来の減色はまず適切な代表点を求めてから，色の分類を始めるのですが，それはこの次の章でやります。まずは人間が使う色を指定してあげましょう。そのほかの，先ほどにはなかったメンバー関数やオペレータ定義は，代表点の自動作成に使うものです。いまは無視してください。

同じ対象でも特徴ベクトルのとり方を変えるとどうなるか，という例でYCbCrの構造体を定義しました。こちらは輝度，色差（青），色差（赤）でひとつの色を表現します。RGBとの違いは簡単にコラムに書いておいたので参照してください。

表３　代表点を与えた減色

元画像	RGBで識別	YCbCrで識別

イメージを取り扱うクラスも定義します。ファイルの入出力とRGB, YCbCr間のコンバート関数で成り立っています。ここで扱うファイルフォーマットはpnm形式です。UNIX では標準的に使われるベタフォーマットで，対応ツールも多く出ています。なによりファイル構造が単純でこういう実験には最適です。pnm形式は，ビットマップを扱うpbm，グレイスケールを扱うpgm，カラーを扱うppmの3種があります。ここではppm形式を使います。ppm形式はヘッダに，マジックナンバー"P6"，横方向サイズ，縦方向サイズ，階調数がテキストで書いてあり，あとはピクセル数分のRGB値がバイナリで羅列されています[*1]。

クラスImageはppmファイルを読んでRGBの配列に取り込むメンバー関数read_ppm()と，逆にRGBの配列からppmファイルに書き出すメンバー関数write_ppm()を備えています。もし手元の処理系にppmを扱う環境がない場合，この2つの関数を書き換えてください。

減色処理のメインルーチンはsupervised_color.ccです。識別器に与える代表点として，RGBのほうは昔ながらのデジタル8色を，YCbCrのほうは赤に少し振った濃淡のグラデーションを用意しました。どちらで識別するかはコマンドラインの3個目の引数で指定します。

コンパイルは,

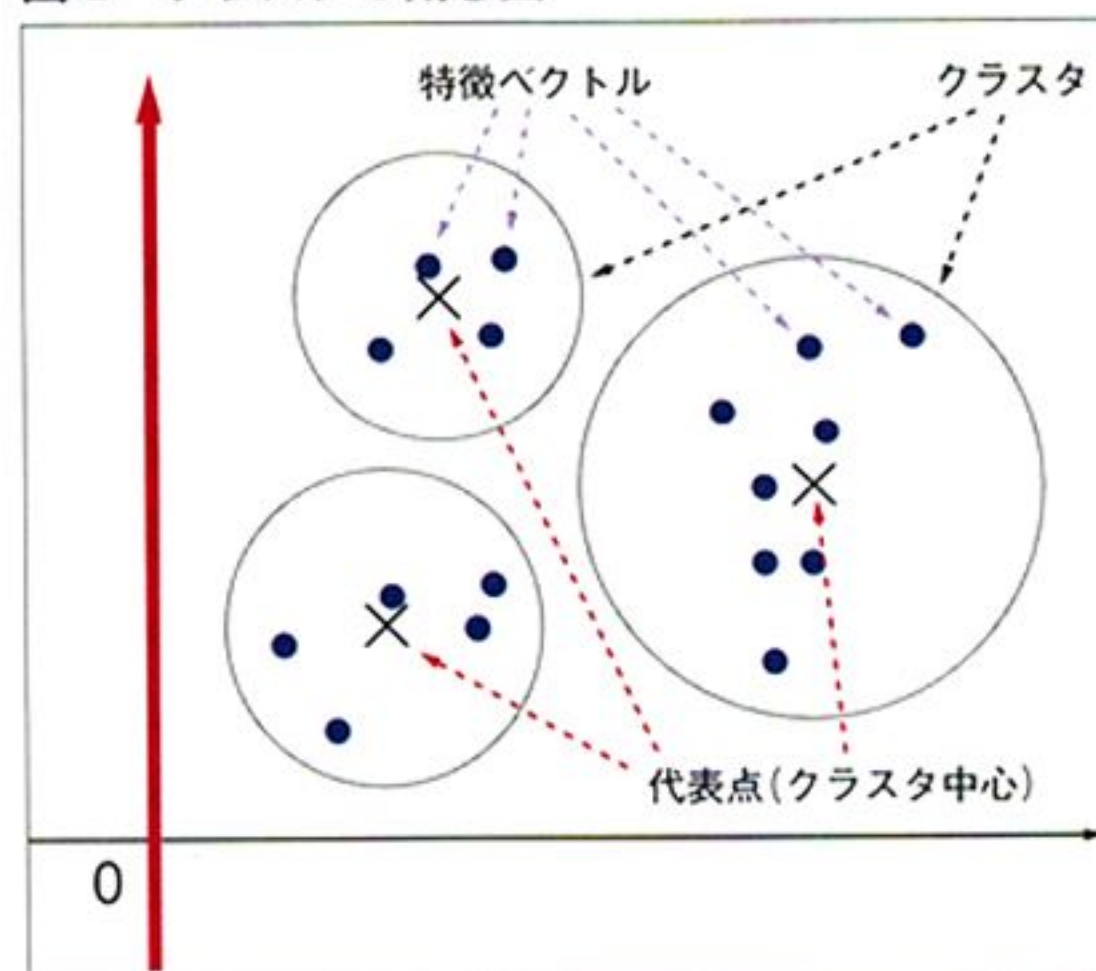
```
%gcc image.cc supervised_color.cc -lm
```

です。sqrt()用にmathライブラリを指定してください。無事通ったら,

```
%a.out source.ppm destination.ppm rgb
```

または,

```
%a.out source.ppm destination.ppm ybr
```

として変換します。

実写とCGという系統の違う2枚の絵で実験した結果を表3に示します。

見事フルカラーの絵が，デジタル8色と，11段階の濃淡画像に変換されました。これはフルカラー，つまり約1677万通りの色のパターンが，8通りと11通りの色のカテゴリに分類されたことを示します。その分類されたカテゴリと代表点テーブルから再構成された画像が，実験結果として出力されたわけです。

さて，この場合のRGBとYCbCrの差は，特徴ベクトルの性質の差というよりも用意された代表点テーブルの差です。RGB側で赤っぽい濃淡のテーブルを用意してやる，もしくは YCbCr側

図3 クラスタの概念図

図4 同一カテゴリが距離的に遠い場合。複数クラスタをひとつのカテゴリに割り当てるが，この図くらいひどいと特徴ベクトルや距離尺度を見直したほうがよい

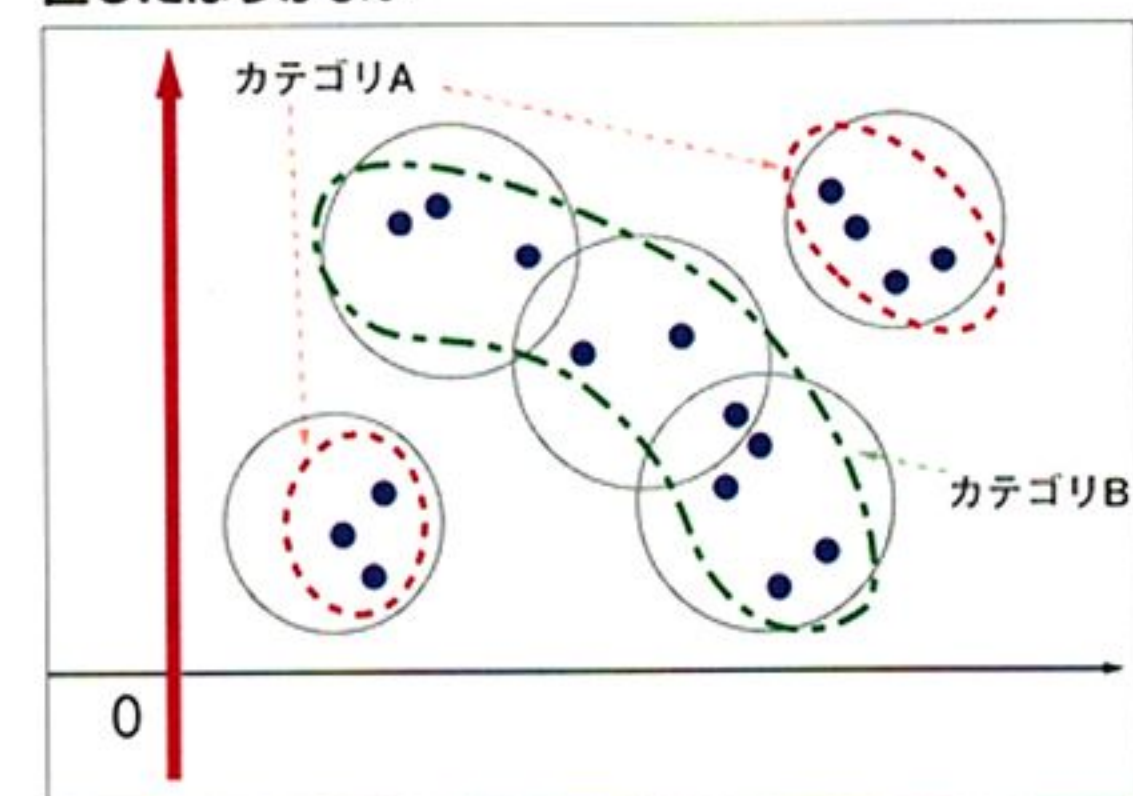



表4 K平均法で16色に自動減色

初期代表点で識別(RGB,16色)	K平均法後に識別(RGB,16色)	K平均法後に識別(YCbCr,16色)

でデジタル8色のテーブルを用意してやれば，ほぼ同じ結果が出るはずです。ただそれを指定するのは直観的にはいかないのですが……。コラムに書いてあるとおり，表色系によって表現しやすい色/しにくい色があるので，自分の意図した効果はどちらのほうが表現しやすいのか考えて，テーブルを作るとよいと思います。

＊2　これをクレヨン12色といいます

＊3　距離尺度はユークリッド距離以外にもいくつも存在します。ただ通常はこれひとつで押してほぼ問題ないでしょう

＊4　テキストで書くバージョンも存在します。マジックナンバーは pbm,pgm,ppm のテキストバージョンが "P1", "P2", "P3" で，バイナリバージョンが "P4", "P5", "P6" です。また，サンプルのソースは簡易版で正確に ppm のフォーマットに沿ったものではありません。コメントやホワイトスペースの挿入状況によっては読めないことがあることをご了承ください。詳しくは % man ppm をどうぞ

■代表点を自動作成してみるのだ

代表点のテーブルを手でいじってみるとときどき思いもよらぬ絵になって面白いのですが，できるだけ元の絵に忠実な代表点を選ぼうとするとけっこう面倒です。そこで代表点を自動作成させてみましょう。

ここでクラスタを正しく説明します。いままでクラスタはカテゴリと同じように考えてきましたが，正しくは特徴空間内でパターン分布が密になっている部分を指します（図3）。分布が密になっているということは（特徴量を適切に選んでいる限り）性質が似通っているものが多く集まっていることになります。それでいままでクラスタとカテゴリを同一視してきたのでした。実際ひとつのカテゴリはひとつないしは数個のクラスタで構成します（図4）。

減色処理の例だと，元画像から色の分布を調べ，色の特徴ベクトルが密になっている空間を探して切り分けると，それがクラスタになります。そして各クラスタの中心が減色処理の代表点となります。このような空間の切り分けをクラスタリングと呼んでいます。

クラスタリングの手法は何種類か提案されていますが，ここではポピュラーな（と思う）「K平均法」を紹介します。筆者はいままでほとんどこれでコトが足りてるので，とりあえずマスターして損はないでしょう。

K平均法の流れは次のようになります。

1）K個の初期代表点を適当に選びクラスタ中心とする
2）与えられた特徴ベクトルのすべてに対し，もっとも近いクラスタを選んで所属させる

リスト3　supervised_color.cc

```
/*
   supervised_color.cc
   代表点を与えられて、減色を行う。
*/

#include <stdio.h>
#include <math.h>
#include "cluster.h"
#include "image.h"

//======== RGB ========//
double RGB::start_dist = 99999.9;
double RGB::break_dist = 0.01;

// RGB の距離関数
double get_dist(const RGB& c0, const RGB& c1)
{
  return sqrt( (c0.r-c1.r)*(c0.r-c1.r)
              +(c0.g-c1.g)*(c0.g-c1.g)
              +(c0.b-c1.b)*(c0.b-c1.b)
              );
}

//======== Ybr444 ========//
double Ybr444::start_dist = 99999.9;
double Ybr444::break_dist = 0.01;

// YCbCr の距離関数
double get_dist(const Ybr444& c0, const Ybr444& c1)
{
  return sqrt( 1.0*(c0.y-c1.y)*(c0.y-c1.y)
              +0.25*(c0.b-c1.b)*(c0.b-c1.b)      // 色情報の重みを
              +0.25*(c0.r-c1.r)*(c0.r-c1.r)      // 小さくする
              );
}

//======== ========//
RGB proto_rgb[] = {                     // デジタル8色の RGB テーブル
  { 0.0, 0.0, 0.0 },
  { 1.0, 0.0, 0.0 },
  { 0.0, 1.0, 0.0 },
  { 1.0, 1.0, 0.0 },
  { 0.0, 0.0, 1.0 },
  { 1.0, 0.0, 1.0 },
  { 0.0, 1.0, 1.0 },
  { 1.0, 1.0, 1.0 },
  { EOD, 0, 0},
};

Ybr444 proto_ybr[] = {                  // 赤グラデーションの YCbCrテーブル
  { 1.0, 0.0, 0.2 },
  { 0.9, 0.0, 0.2 },
  { 0.8, 0.0, 0.2 },
  { 0.7, 0.0, 0.2 },
  { 0.6, 0.0, 0.2 },
  { 0.5, 0.0, 0.2 },
  { 0.4, 0.0, 0.2 },
  { 0.3, 0.0, 0.2 },
  { 0.2, 0.0, 0.2 },
  { 0.1, 0.0, 0.2 },
  { 0.0, 0.0, 0.2 },
  { EOD, 0, 0},
};

//======== メイン ========//
int main(int argc, char *argv[])
{
  Cluster<RGB>            rgb_clst(proto_rgb);
  Cluster<Ybr444> ybr_clst(proto_ybr);
  Image      img;

  if (argc < 3) errorm( "usage: %%%s input.ppm output.ppm [r(gb)/y(br)]¥n" ,argv[0]);
  img.read_ppm(argv[1]);
  if ((argc == 3) || ((argc > 3) && (*argv[3] == 'r' ))) {
    printf( "RGB¥n" );
    rgb_clst.discriminate(img.rgb, img.rgb, NULL);       // 減色
  } else {
    printf( "YCbCr¥n" );
    img.RGBtoYbr444();
    ybr_clst.discriminate(img.ybr, img.ybr, NULL);       // 減色
    img.Ybr444toRGB();
  }
  img.write_ppm(argv[2]);
}
```

リスト4　k_mean_color.cc

```
/*
   k_mean_color.cc
   K平均法を使って減色処理をする。
*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "cluster2.h"
#include "image.h"

//======== RGB ========//
double RGB::start_dist = 99999.9;
double RGB::break_dist = 0.01;

double get_dist(const RGB& c0, const RGB& c1)
{
  return sqrt( (c0.r-c1.r)*(c0.r-c1.r)
              +(c0.g-c1.g)*(c0.g-c1.g)
              +(c0.b-c1.b)*(c0.b-c1.b)
              );
}

//======== Ybr444 ========//
double Ybr444::start_dist = 99999.9;
double Ybr444::break_dist = 0.01;

double get_dist(const Ybr444& c0, const Ybr444& c1)
{
  return sqrt( 1.0*(c0.y-c1.y)*(c0.y-c1.y)
              +0.25*(c0.b-c1.b)*(c0.b-c1.b)
              +0.25*(c0.r-c1.r)*(c0.r-c1.r)
              );
}

//======== メイン ========//
int main(int argc, char *argv[])
{
  Image      img;
  int   ncol,loop;

  if (argc < 4) errorm( "usage: %%%s input.ppm output.ppm num_color [r(gb)/y(br)]ÅñYn" ,argv[0]);
  ncol = atoi(argv[3]);                  // 色数
  loop = 6;                              // クラスタリングループ回数
  img.read_ppm(argv[1]);

  if ((argc == 4) || ((argc > 4) && (*argv[4] == 'r' ))) {
    // RGB
    printf( "RGB\nselecting initial prototypes...¥n" );
    Cluster<RGB>  rgb_clst(ncol,img.rgb);    // 初期代表点を適当に選んでもらう
    printf( "clustering by k-mean method...¥n" );
    rgb_clst.k_mean(img.rgb, loop);                      // クラスタリング
    printf( "discriminating...¥n" );
    rgb_clst.discriminate(img.rgb, img.rgb, NULL);       // 減色
  } else {
    // YCbCr
    img.RGBtoYbr444();
    printf( "YCrCr\nselecting initial prototypes...¥n" );
    Cluster<Ybr444>    ybr_clst(ncol,img.ybr);
    printf( "clustering by k-mean method...¥n" );
    ybr_clst.k_mean(img.ybr, loop);
    printf( "discriminating...¥n" );
    ybr_clst.discriminate(img.ybr, img.ybr, NULL);
    img.Ybr444toRGB();
  }
 printf( "done.¥n" );
 img.write_ppm(argv[2]);
}
```

3）各クラスタに所属している特徴ベクトルの平均をとって，新しいクラスタ中心にする
4）クラスタ中心が動かなくなるまで 2），3）の処理を繰り返す

これだけです。長い前フリだっただけに拍子抜けでしょうか。要は識別結果から新しい代表点を作ることの繰り返しです。

K平均法の処理を組み込んだのがcluster2.hです。与えられた特徴ベクトルの配列から適当にK個選んで代表点にするコンストラクタが追加されています。もちろんK平均法でクラスタリングするメンバー関数も追加です。新しい代表点を求めるのに特徴ベクトルの加算や乗算が必要になるので，その構造体のほうでオペレータの再定義を行うようにします。先ほどimage.hのなかでRGBやYCbCrのオペレータを定義していたのはこのためです。

表5　K平均法で256色に自動減色

K平均法後に識別識別（YCbCr,256色）

K平均法は，本来はクラスタ中心が動かなくなるまで，つまり中心の移動距離が一定値以下になるまで修正を繰り返します。しかし最初からその閾値を出すのは大変なので，ループ回数を与えてその回数だけ計算するようにしています。1ループ計算するたびにクラスタ中心の移動距離の平均を出すようにしていますから，その値を参考に打ち切り回数を定めてください。

cluster2.hのK平均法を使って自動減色処理を行うプログラムをk_mean_color.ccに示します。変数 loop に計算ループ回数が入っています。これを0にするとK平均法によるクラスタリングを行わず，初期代表点で識別します。使用前後の比較用です。

では，画像を自動減色しましょう。コンパイルと実行は，

```
%gcc image.cc k_mean_color.cc -lm
%a.out source.ppm destination.ppm num_
color [r(gb)/y(br)]
```

です。24ビットフルカラーから16色まで減色した結果を表4に示します。

K平均法によるクラスタリングの効果がわかると思います。適当に選ばれた初期代表点から，より最適に近い代表点が選ばれています。

また，RGBとYCbCrではコントラストの再現度合に差が出ていることがわかるでしょうか？YCbCrのほうが元画像に近いです。これは YCbCrの距離関数で，輝度の係数に重みをつけているためです。CbCrの係数を0にして完全に輝度のみで距離を測るとモノトーンになりますし，逆に色を重視するならCbCrの係数を大きくします。RGBのほうだと青を重視とか赤を重視とかの指定ができるでしょう。このように距離関数で識別器の振る舞いががらりと変わります。いろいろと重みをいじってみると面白いでしょう。

16色はさすがに劣化が厳しいので，256色に減色した画像も作ってみました（表5）。

これくらいの劣化だと十分実用範囲内です。

ところでK平均法の計算コストは，対象特徴ベクトル数×クラスタ数×ループ数になります。仮に640×480の画像を256色に減色，ループ数を10回とすると，640×480×256×10＝786,432,000回の距離計算が必要です。最近のマシンは速いのでこれくらいまともに計算しても少々待てば終わってしまいますが[*5]，早く終わるに越したことはありません。すでに工夫のひとつとして break_dist 以下になったらそのクラスタに属するとみなして，以降の距離計算を打ち切るようにしています。ただ打ち切りによってクラスタリングの誤差が出ることが難点です。上の例だと実写のほうは問題ありませんが，CGのほうはグラデーションにマッハバンド（グラデーションの段階）が出ています（リボンより手足部分のほうがわかりやすい）。これは色のクラスタ数が足りなかったのではなく，打ち切りによってグラデーションの微妙な輝度変化を識別できなくなったためです。かといってあまり小さいと計算量を減らせません。対象にあわせた閾値設定が大切です。

もうひとつ筆者がよくやる手は，画像の空間コヒーレンス（距離が近いと相関が高いこと）を期待して1回のクラスタリングで扱う特徴ベクトルを適当にスキップ（たとえば4ピクセル置き）させることです。画像，特に動画は本当に量が多いので計算量を抑える工夫が欠かせません。

＊5　この実験は愛機 Let's note（MMX 150MHz, 96MB）でやってます。こんなに小さいのに本当に速く計算します。昔，SUN3をぶん回してクソデカイ固有値計算してた頃が夢のよう。しかしO2（R10000 174MHz, 256MB）でやると Let's note の3倍くらい速い……

■画像圧縮するのだ

最後に画像圧縮に取り組んでみましょう。

すでに述べたとおり，パターン認識は情報量削減の側面を持ち，分類されたカテゴリと代表点テーブルから，元の情報に近いものを復元することができます。実際，減色処理はそのまま画像の圧縮になっています。フルカラーだと1ピクセルを表現するのにRGB各8ビット，計24ビット必要でしたが，色分類後のカテゴリを保存するようにすれば，256色なら8ビットですみます。クラスタ中心の保存分も必要ですが，この時点で約1/3の圧縮になります。

図5　2×2ピクセルをひとまとめに扱う

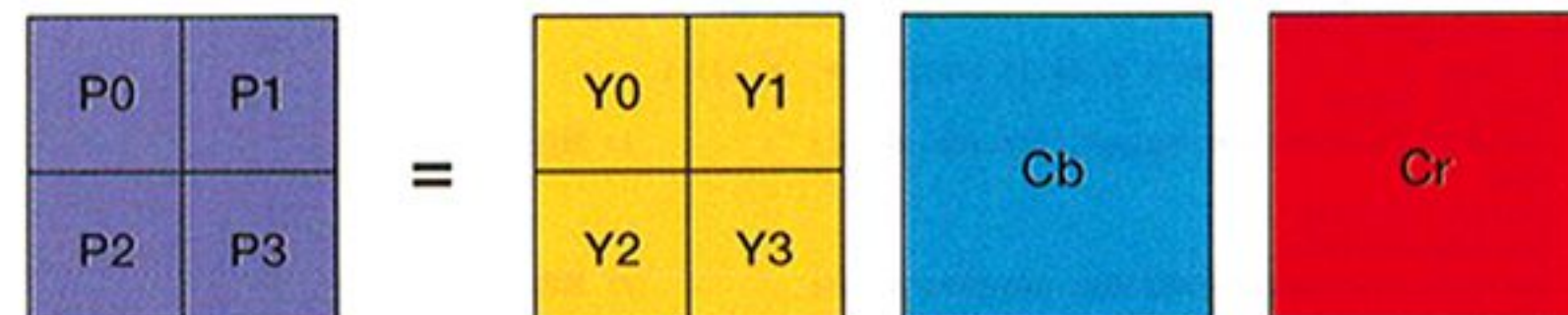
2×2のピクセルは各々4つずつのRGB要素から成り立っている…

P0 P1 / P2 P3 ＝ R0 R1 / R2 R3　G0 G1 / G2 G3　B0 B1 / B2 B3

つまり次のような12次元の特徴ベクトルで表すことが可能
（R0, G0, B0, R1, G1, B1, R2, G2, B2, R3, G3, B3）

図6　CbCrは4ピクセルでひとつにする

CbCrは4つまとめてひとつにしてしまう

P0 P1 / P2 P3 ＝ Y0 Y1 / Y2 Y3　Cb　Cr

これで6次元のベクトルになる
（Y0, Y1, Y2, Y3, Cb, Cr）

もうひと声圧縮率を上げてみましょう。16色の画像を見てもわかるとおりひとつのピクセルを潰すのは限度がありますから，複数のピクセルをまとめてひとつの特徴ベクトルとして扱ってみます。たとえば，2×2の4ピクセルをひとまとめにすると，特徴ベクトルは図5のように表せます。

こうすると1画面当たりの入力パターン数は1/4になります。これを256個のクラスタに分類すると，4ピクセル当たり8ビットで表現できます。元は8×3×4＝96ビットなので約1/12の圧縮です。

このまま識別器に放り込んでもいいのですが，4ピクセルのRGB値を並べた12次元のベクトルは少し大きいので，YCbCrのほうを使ってCbCr成分を4ピクセル分まとめてひとつにしてしまいます（図6）。

人間の目は輝度に敏感で色に鈍感なことを利用した形式です（このような形式をYCbCr 4:2:0フォーマットといいます）。これで輝度4，色差2の6次元に減らすことができました。

あとはこれまでと同じように特徴ベクトルの構造体を組んでやるだけです。k_mean_image.ccに実装を示します。Ybr420_2という構造体がそれです。4個のピクセルを一度に表現できる特徴ベクトルを作り，そのオペレータや距離関数を定義しています。6次元になっても距離の求め方は同じです。多次元空間はイメージしにくいですが，距離関数を見ればいままでとまったく同じ手法を用いていることがわかるでしょう。

RGBとの変換も必要なので，クラスImageを継承してImage2というYCbCr 4:2:0も扱えるクラスを作りました。

おまけに8×8ピクセルをひとまとめにしたYbr420_8も追加しました。この特徴ベクトルは，

8×8（Y）＋4×4（(b)＋4×4（Cr）＝96次元

の大きなベクトルです（図7）。

いままでとは桁が違う大きさで，こんなのありか？　と思ってしまうかもしれませんね。無論ありです。要は距離さえ求まれば問題ありません。さて圧縮率は素直に，

1/3×1/64＝1/192（！）

……といきますかどうか。

図7　8×8をまとめた特徴ベクトル

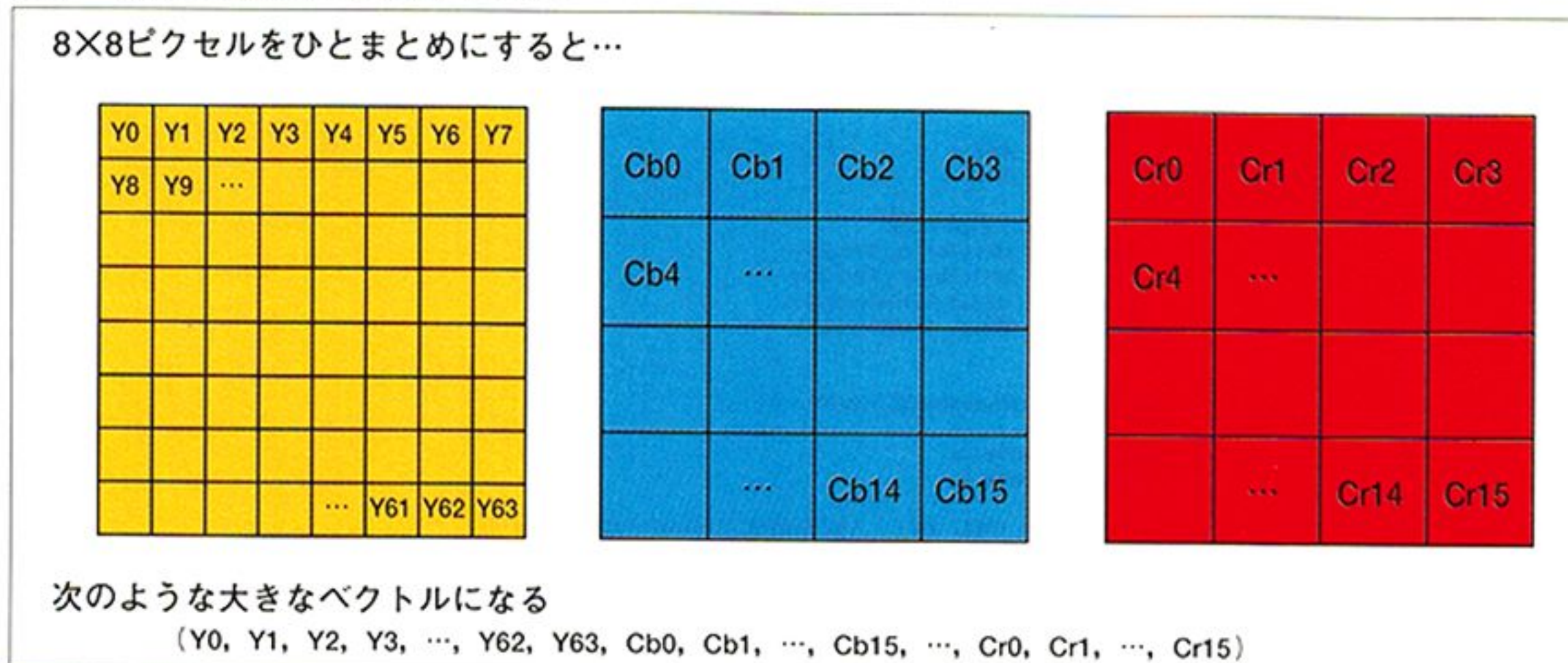
8×8ピクセルをひとまとめにすると…

Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
Y8	Y9	…					
				…	Y61	Y62	Y63

Cb0	Cb1	Cb2	Cb3
Cb4	…		
	…	Cb14	Cb15

Cr0	Cr1	Cr2	Cr3
Cr4	…		
	…	Cr14	Cr15

次のような大きなベクトルになる

（Y0，Y1，Y2，Y3，…，Y62，Y63，Cb0，Cb1，…，Cb15，…，Cr0，Cr1，…，Cr15）

リスト5　k_mean_image.cc

```
/*
   k_mean_image.cc
   K平均法を使って画像圧縮をする。
*/

#include <stdio.h>
#include <stdarg.h>
#include <math.h>
#include <stdlib.h>
#include "cluster2.h"
#include "image.h"

//======== Ybr444 ========//
double Ybr444::start_dist = 99999.9;
double Ybr444::break_dist = 0.01;

double get_dist(const Ybr444& c0, const Ybr444& c1)
{
  return sqrt( 1.0*(c0.y-c1.y)*(c0.y-c1.y)
                +0.25*(c0.b-c1.b)*(c0.b-c1.b)
                +0.25*(c0.r-c1.r)*(c0.r-c1.r)
                );
}

//======== YCbCr420(2x2) ========//
struct Ybr420_2 {
  double y[4],b,r;
  static double start_dist;
  static double break_dist;
  Ybr420_2& operator+=(const Ybr420_2&);
  Ybr420_2& operator*=(const double);
  void      toYbr420_2(const RGB *rgb, int ix);
  void      toRGB_420_2(RGB *rgb,int ix);
  inline int is_end(void) {
    if (y[0] == EOD) return -1;
    return 0;
  }
  inline void end(void) {
    y[0] = EOD;
  }
  void      zero(void) {
    y[0] = y[1] = y[2] = y[3] = b = r = 0;
  }
  void      print(void){};
};

double Ybr420_2::start_dist = 99999.9;
double Ybr420_2::break_dist = 0.01;

Ybr420_2& Ybr420_2::operator+=(const Ybr420_2& c)
{
  for (int i=0; i<4; i++) {
    y[i] += c.y[i];
  }
  b += c.b;
  r += c.r;
  return *this;
}

Ybr420_2& Ybr420_2::operator*=(const double m)
{
  for (int i=0; i<4; i++) {
    y[i] *= m;
  }
  b *= m;
  r *= m;
  return *this;
}

Ybr420_2 operator*(const Ybr420_2& c, double m)
{
  Ybr420_2 c0 = c;
  c0 *= m;
  return c0;
}

double get_dist(const Ybr420_2& c0, const Ybr420_2& c1)
{
  double d = 0;
  for (int i=0; i<4; i++) {
    d += (c0.y[i] - c1.y[i])*(c0.y[i] - c1.y[i]);
  }
  d += (c0.b - c1.b)*(c0.b - c1.b);
  d += (c0.r - c1.r)*(c0.r - c1.r);
  return sqrt(d);
}

// RGB 4ピクセルから YCbCr 4:2:0 に変換する
void Ybr420_2::toYbr420_2(const RGB *rgb, int ix)
{
  double tb,tr;
  Ybr444 tc;

  tc = toYbr444(rgb[0]);
  y[0] = tc.y;
  tb = tc.b;
  tr = tc.r;

  tc = toYbr444(rgb[1]);
  y[1] = tc.y;
  tb += tc.b;
  tr += tc.r;

  tc = toYbr444(rgb[ix]);
  y[2] = tc.y;
  tb += tc.b;
  tr += tc.r;

  tc = toYbr444(rgb[ix+1]);
  y[3] = tc.y;
  tb += tc.b;
  tr += tc.r;

  b = tb/4.0;
  r = tr/4.0;
}

// YCbCr 4:2:0 から RGB 4ピクセルに変換する
void Ybr420_2::toRGB_420_2(RGB *rgb,int ix)
{
  Ybr444 tc;

  tc.b = b;
  tc.r = r;

  tc.y = y[0];
  rgb[0] = toRGB(tc);

  tc.y = y[1];
  rgb[1] = toRGB(tc);

  tc.y = y[2];
  rgb[ix] = toRGB(tc);
```

```
  tc.y = y[3];
  rgb[ix+1] = toRGB(tc);
}

//======== YCbCr420(8x8) ========//
struct Ybr420_8 {
  double y[64],b[16],r[16];
  static double start_dist;
  static double break_dist;
  Ybr420_8& operator+=(const Ybr420_8&);
  Ybr420_8& operator *=(const double);
  void       toYbr420_8(const RGB *rgb, int ix);
  void       toRGB_420_8(RGB *rgb,int ix);
  inline int is_end(void) {
    if (y[0] == EOD) return -1;
    return 0;
  }
  inline void end(void) {
    y[0] = EOD;
  }
  void       zero(void);
  void       print(void){};
};

double Ybr420_8::start_dist = 99999.9;
double Ybr420_8::break_dist = 0.03;

void Ybr420_8::zero(void)
{
  int        i;
  for (i=0; i<64; i++) {
    y[i] = 0;
  }
  for (i=0; i<16; i++) {
    b[i] = r[i] = 0;
  }
}

Ybr420_8& Ybr420_8::operator+=(const Ybr420_8& c)
{
  int        i;
  for (i=0; i<64; i++) {
    y[i] += c.y[i];
  }
  for (i=0; i<16; i++) {
    b[i] += c.b[i];
    r[i] += c.r[i];
  }
  return *this;
}

Ybr420_8& Ybr420_8::operator *=(const double m)
{
  int        i;
  for (i=0; i<64; i++) {
    y[i] *= m;
  }
  for (i=0; i<16; i++) {
    b[i] *= m;
    r[i] *= m;
  }
  return *this;
}

Ybr420_8 operator *(const Ybr420_8& c, double m)
{
  Ybr420_8 c0 = c;
  c0 *= m;
  return c0;
}

double get_dist(const Ybr420_8& c0, const Ybr420_8& c1)
{
  double d = 0;
  int        i;
  for (i=0; i<64; i++) {
    d += (c0.y[i] - c1.y[i])*(c0.y[i] - c1.y[i]);
  }
  for (i=0; i<16; i++) {
    d += (c0.b[i] - c1.b[i])*(c0.b[i] - c1.b[i]);
    d += (c0.r[i] - c1.r[i])*(c0.r[i] - c1.r[i]);
  }
  return sqrt(d);
}

// RGB 64ピクセルから YCbCr 4:2:0 に変換する
void Ybr420_8::toYbr420_8(const RGB *rgb, int ix)
{
  int  i,j;
  double tb,tr;
  Ybr444 tc;

  for (i=0; i<8; i+=2) {
    for (j=0; j<8; j+=2) {
      tc = toYbr444(rgb[i*ix+j]);
      y[i*8+j] = tc.y;
      tb = tc.b;
      tr = tc.r;

      tc = toYbr444(rgb[i*ix+j+1]);
      y[i*8+j+1] = tc.y;
      tb += tc.b;
      tr += tc.r;

      tc = toYbr444(rgb[(i+1)*ix+j]);
      y[(i+1)*8+j] = tc.y;
      tb += tc.b;
      tr += tc.r;

      tc = toYbr444(rgb[(i+1)*ix+j+1]);
      y[(i+1)*8+j+1] = tc.y;
      tb += tc.b;
      tr += tc.r;

      b[(i/2)*(8/2)+j/2] = tb/4.0;
      r[(i/2)*(8/2)+j/2] = tr/4.0;
    }
  }
}

// YCbCr 4:2:0 から RGB 64ピクセルに変換する
void Ybr420_8::toRGB_420_8(RGB *rgb,int ix)
{
  int        i,j;
  Ybr444 tc;

  for (i=0; i<8; i+=2) {
    for (j=0; j<8; j+=2) {
      tc.b = b[(i/2)*(8/2)+j/2];
      tc.r = r[(i/2)*(8/2)+j/2];

      tc.y = y[i*8+j];
      rgb[i*ix+j] = toRGB(tc);

      tc.y = y[i*8+j+1];
      rgb[i*ix+j+1] = toRGB(tc);

      tc.y = y[(i+1)*8+j];
      rgb[(i+1)*ix+j] = toRGB(tc);

      tc.y = y[(i+1)*8+j+1];
      rgb[(i+1)*ix+j+1] = toRGB(tc);
    }
  }
}

//======== YCbCr 4:2:0 をサポートした Image クラス ========//
class  Image2:public Image {
protected:
  int        xsize2, ysize2;
  int        xsize8, ysize8;
public:
  Ybr420_2  *ybr2;
  Ybr420_8  *ybr8;
  Image2():Image() {
    ybr2 = NULL;
    ybr8 = NULL;
  }
  -Image2() {
    delete[] ybr2;
    delete[] ybr8;
  }
  // それぞれの一画面分の変換
  void       RGBtoYbr420_2(void);
  void       RGBtoYbr420_8(void);
  void       Ybr420_2toRGB(void);
  void       Ybr420_8toRGB(void);
};

void Image2::RGBtoYbr420_2(void)
{
  xsize2 = xsize/2;
  ysize2 = ysize/2;

  if (ybr2 == NULL) ybr2 = new Ybr420_2[xsize2*ysize2+1];
  for (int i=0; i<ysize2; i++) {
    for (int j=0; j<xsize2; j++) {
      ybr2[i*xsize2+j].toYbr420_2(&rgb[(i*xsize+j)*2],xsize);
    }
  }
  ybr2[xsize2*ysize2].end();
}

void Image2::RGBtoYbr420_8(void)
{
  xsize8 = xsize/8;
  ysize8 = ysize/8;

  if (ybr8 == NULL) ybr8 = new Ybr420_8[xsize8*ysize8+1];
  for (int i=0; i<ysize8; i++) {
    for (int j=0; j<xsize8; j++) {
      ybr8[i*xsize8+j].toYbr420_8(&rgb[(i*xsize+j)*8],xsize);
    }
  }
  ybr8[xsize8*ysize8].end();
}

void Image2::Ybr420_2toRGB(void)
{
  for (int i=0; i<ysize2; i++) {
    for (int j=0; j<xsize2; j++) {
      ybr2[i*xsize2+j].toRGB_420_2(&rgb[(i*xsize+j)*2],xsize);
    }
  }
}

void Image2::Ybr420_8toRGB(void)
{
  for (int i=0; i<ysize8; i++) {
    for (int j=0; j<xsize8; j++) {
      ybr8[i*xsize8+j].toRGB_420_8(&rgb[(i*xsize+j)*8],xsize);
    }
  }
}

//======== メイン ========//
int main(int argc, char *argv[])
{
  Image2 img;
  int   ncell,loop;

  if (argc < 4) errorm( "usage: %%%s input.ppm output.ppm num_cell [2/8]¥n" ,argv[0]);
  ncell = atoi(argv[3]);
printf( "ncell %d¥n" ,ncell);
  loop = 4;
  img.read_ppm(argv[1]);

  if ((argc == 4) || ((argc > 4) && (*argv[4] ==  '2' ))) {
    // 2x2
    img.RGBtoYbr420_2();
    printf( "YCbCr420(2x2)¥nselecting initial prototypes...¥n" );
    Cluster<Ybr420_2>  ybr2_clst(ncell,img.ybr2);
    printf( "clustering by k-mean method...¥n" );
    ybr2_clst.k_mean(img.ybr2, loop);
    printf( "discriminating...¥n" );
    ybr2_clst.discriminate(img.ybr2, img.ybr2, NULL);
    img.Ybr420_2toRGB();
  } else {
    // 4x4
    img.RGBtoYbr420_8();
    printf( "YCrCr420(8x8)¥nselecting initial prototypes...¥n" );
    Cluster<Ybr420_8>    ybr8_clst(ncell,img.ybr8);
    printf( "clustering by k-mean method...¥n" );
    ybr8_clst.k_mean(img.ybr8, loop);
    printf( "discriminating...¥n" );
    ybr8_clst.discriminate(img.ybr8, img.ybr8, NULL);
    img.Ybr420_8toRGB();
  }
  printf( "done.¥n" );
  img.write_ppm(argv[2]);
}
```

コンパイル方法と実行は次のとおりです。

```
%gcc image.cc k_mean_image.cc -lm
%a.out source.ppm destination.ppm num_cell [2/8]
```

結果を表6に示します。

2×2ピクセルをまとめたほうは，まあまあ見られる画質です。拡大すると2×2のブロックが繰り返されている部分が確認できます。CGのほうを見ると，白との境界線は割と綺麗なのに，赤との境界線はぼやっとしています。色成分をまとめて重みを小さくした影響がよくわかります。

8×8ピクセルをまとめたほうはさすがに無理があったようです。まさにモザイクをかけた状態。確かに8×8の単位で代表パターンを使い回すわけですから，元画像の再構成は厳しいでしょう。1/192の圧縮率は夢でした(情報量自体は縮小されている)。クラスタ数を増やせばなんとかなりそうですが，クラスタ中心を保存する容量が馬鹿にならなくなります。なにしろベタのRGB(192次元)の半分もありますから。一応これを減らすことはできます。しかし今度は各ブロックごとに次元数を減らすことがメインになり，クラスタリングは無用になってしまうので本稿では割愛します*6。

さて，クラスタリングやパターン分類には随分時間がかかりますが，逆に分類済みのカテゴリから属するクラスタ中心を引っ張り出すのは簡単です。配列を読むだけですから。アルゴリズムが単純なだけにハードウェア化も容易で，リアルタイム性を必要とする展開に向いています。これが信号処理などでベクトル量子化(VQ)と呼ばれる圧縮法です。もちろんこのネーミングはベクトルをスカラー量(カテゴリ)に量子化している，という見方からきています。

*6　このブロック単位にDCT(離散コサイン変換)をかけて次元数を圧倒的に減らすのが，JPEGの圧縮アルゴリズムです。DCTはコラムに書いてあるKLTに近い特性を持つことが知られています

■オチなのだ

減色にしてもVQ圧縮(特に8×8のほう)にしても，昔どこかで見たような……という既視感にとらわれた人，きっといると思います。前者はパレットカラーだし，後者はPCGそのもの。

ハイ，正解です。

その昔，時代は8ビット機でまだマシンパワーのなかった頃。大量のフレームバッファを積む代わりに，こうやって情報削減して頑張って絵を出してきたのです。当時筆者もその理論的背景などカケラも考えずに頑張っていたひとりでした。いま，こうして眺め直してみると，なんとなくあの時代がいとおしく感じられませんか？

表6　複数ピクセルをまとめた画像圧縮

2×2(クラスタ数256)	8×8(クラスタ数256)

表色系について

COLUMN

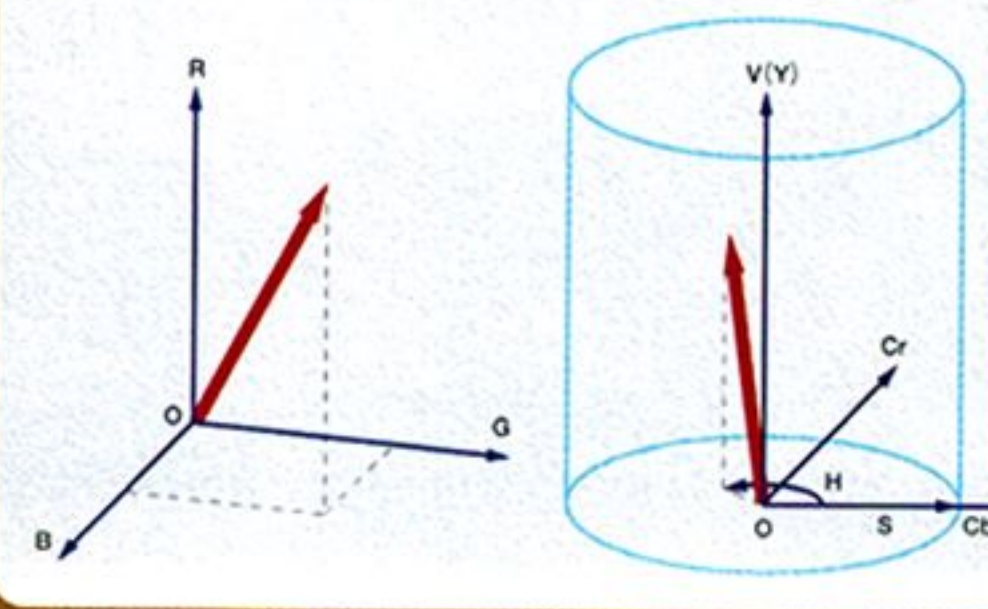
コンピュータで色を表すのには，通常光の3原色を使ったRGB表色系が用いられます。もうひとつよく用いられるのが色相，明度(輝度)，彩度を使った表色系です。グラフィックツールなどで，HSV(Hue,Saturation,Value)とかHVC(Hue,Value,Chroma)という色の表現を見たことがあると思います。

RGB表色系は直角座標系です。それに対し，HSVなどは円柱座標系になります(実際には輝度の両端で色の存在範囲が狭まるので，使える空間は紡錘形になる)。本文中で用いているYCbCrは輝度と色差といわれるものです。HSVとの対応は，YがV，(Cb,Cr)の長さがS，(Cb,Cr)の角度がH，となります。

RGB表色系は，たとえば赤が強かった場合，ほかの緑や青も強くなる傾向があります。このように各要素の関わりが強いことを相関が高いといいます。HSV表色系ではひとつの物体に光が当たっていて輝度変化の激しい部分でも，その物体内の色相や彩度はあまり変化しません。相関が低いわけです。また，本の判型の縦横の長さは非常に相関の高い例です。相関が高い＝冗長な表現であるため，パターン認識に限らず統計などの分野でも，まず相関が低い表現に変換し特徴ベクトルの次元数を低減させてから解析にかかります。この作業が，主成分分析やKarhunenLoeve変換(KLT)と呼ばれるものです。

HSV表色系では人間の視覚特性が輝度に敏感で色には鈍感であることから，輝度情報を優先して保存し，色情報を落とすことをよく行います。本文中，YCbCr 4:2:0のように比率を書いていますが，これが輝度と色の保存比率です。4：4：4はYCbCrすべて対等，4：2：2はCbCrが半分，4：2：0はCbCrが1/4になっていることを表します(歴史的経緯で4：1：1でなく4：2：0と書きます)。

ところでRGBに比べHSVなどの表色系のほうが人間に馴染みやすいとよくいわれますが，これは個人の慣れの問題だと筆者は思っています(単に自分が長年RGBで指定してただけという話も)。ただそれぞれに得手不得手があるのは確かで，場合によって使い分けるのがベストです。ちなみに某アーケードゲームのバージョンアップの際に行った画面の色変更は，ほとんどHSVで行いました。

RGB表色系とHSV(YCbCr)表色系

仏日翻訳プログラム「来来仏語」

石上 達也/Ishigami Tatsuya

以前はX68000でmicro EmacsとASK68Kを用いて原稿を書いていましたが，いまはMS Wordでこれを書いています。当時は，あれほど不可能と思われていた，一般ワープロでの文章作成も，いまではそれほど苦でもなくなりました。慣れと勢いとは恐ろしいものです。

Oh!Xが休刊してからの3年間に，なにをやっていたかというのを紹介します。ここではNIFTY SERVEのFLR Forum（外国語フォーラムロマンス語派分館）にアップロードしている仏日翻訳プログラムの話をさせてもらいます。

■個人的な昔話

どういうわけか，昔からアセンブラとかコンパイラという言葉に妙に引かれる癖が私にはありまして，GAME[*1]とかWICS[*2]というプロジェクトを見つつ，いつか自分もそんなプログラムが作りたいなぁと思っていました。この頃は数式をどう処理するかが難題で，

a＝1＋2＊3

をどうやって機械語に翻訳するか，というところで行き詰まっていました。単純に左から処理していくと，

a＝1＋2＊3→(1＋2)＊3

となってしまい，本来の結果と異なってしまいます。また，すべての形について，

A1＋A2＊A3
→reg0＝A2＊A3
reg0＝reg0＋A1

のようなテンプレートを持つわけにもいきません。

ちょうど，そんなとき，瀧山氏のFuzzy BASIC（Oh!MZ 1987年1月号）に出合い，数式処理に関する疑問点は氷解しました。その勢いでFuzzy BASIC Compiler（同1987年6月号）を作り上げました。ラベル，変数のメモリ割り当て，ループ処理の展開などは数式処理で悶々としている間にいくらでも考える時間はありましたから，あとは気合と若さです（編注：投稿時，石上君は16歳）。

これは，私のOh!Xでの出発点でしたが，コンピュータを使った言語処理の第一歩でもありました。

＊1　詳細は忘れてしまったが，1982年頃，月刊ASCIIで盛り上がっていたインタプリタ言語

＊2　同，1979年当時，月刊I/Oで盛り上がっていたコンパイラ言語

■自動翻訳

さぁ，アセンブラは作った（1990年7月号），コンパイラは作った，ウィンドウプログラムも作れるようになった（1994年3月号），というわけで，たいていのプログラム言語は作れるようになりました。このままいくと，最後の難関，自然言語を処理しようということになりますが，いや，やっぱり敷居は高いです。

よく，新聞の切り抜きを自動翻訳プログラムに突っ込んで，あぁ，やっぱり自動翻訳プログラムは使えないのねぇ，という結論を出す人がいますが，それは最初から高望みのしすぎです。

自動翻訳といっても（たとえ，人工知能を搭載してても），翻訳される文章はプログラムが用意したパターン（文法規則）のどれかと一致しなければなりません。つまり，

1）倒置した文

例）Suddenly cried he.
突然，彼は叫んだ。

2）文章の一部が省略された文

例）He has a blue cap and she has white.
彼は青い帽子を持っていて，彼女は白いのを持っている（普通，〝have〟＋形容詞というのは，通常の文法規則ではない。自動翻訳を行うなら，He has a blue cap and she has white one.とすべき）。

3）翻訳するのに予備知識が必要な文

例）He reads papers everyday.
毎日，彼は紙を読む → 毎日，彼は新聞を読む

4）特殊ないい回しが必要な文

例）This instruction pushes the value in the stack.
この命令は，その値をスタックの中に押す。
→この命令は，その値をスタックに積む。

5）単語が1対1で対応していない文

例）He is Japanese.
彼は日本語（Japanese）だ。
→彼は日本人（Japanese）だ。

6）文脈を考慮する必要がある文

例）He could be there at that time.
あのとき，彼はそこに行けた。
あるいは，
あのとき，彼はそこにいたはずだ。

などで，自動翻訳の精度を計ってはかわいそうです。英字新聞は日本人が読んでも難しいのです。内容が理解できれば，文法規則に加えて，〝暗黙の了解〟が，さらに解析パターンに加えられますが，意味がわからなければ，お手上げです。

```
F2J
自動
D:¥xcd2¥F2J>f2j
仏> Le Canada est dis-huit fois plus grand que la France.
日> カナダだ言えフランスより8倍ひどい。
仏> Le Canada est dix-huit fois plus grand que la France.
日> カナダはフランスより18倍大きい。
仏> Je suis un peu plus [g[ que Georges.
日> ジョルジュより少し私は年上だ。
仏> Que vous r}serve la rentr}e ?
There isn't <r}serve> in my dictionary
日> あなた。
仏> Le gris entre dans la danse.
日> (その)灰色だ入れ中に(彼女/それ)を踊られろ。
仏> Un livre a toujours {t{ pour moi un ami, un consolateur {loquent et calme.
日> 本は私にとっていつも友達、雄弁で、静かな慰める人だった。
仏> Que vous r{serve la rentr{e ?
日> 何あなた戻された条件/制限/控え目(彼女/それ)?
仏> Tu feras tes devoirs avec application.
日> 君の宿題を一生懸命にしろ。
仏> Mon avis n'est pas toujours le tien.
日> 私の意見/通知は必ずしも君のものではない。
仏> Toutes les tendances de la rentr{e avec.
日> 全ての傾向戻された(彼女/それ)いっしょだ。
仏> _
```

翻訳の実行結果。うーむ……

■下訳

そんな問題の多い自動翻訳ですが，人間の翻訳を補助させるという使い方に限定すれば，それなりの用途があります。

長文の翻訳をやると，人間の集中力には限度があることを思い知らされます。よし，やるぞー，と気合が入っていても作業の途中で徐々に低下していきます。初めは，これ以上の日本語はないというくらいちゃんとした日本語でも，後ろにいくにつれ，意味はわかるんだけどねぇー，となって，最後には，原文を見ないとなにをいっているのかさっぱりわからん，というレベルまで一気に低下します。後日，翻訳結果を見直していくと，それぞれの個所の集中力が手に取るようにわかるはずです。

外国語を見て，日本語を見て，外国語を見て，日本語を見て，という作業を繰り返し行っているうちに，自分の中の日本語のリズムが外国語に引きずられて，壊れてしまうからではないでしょうか？　「あなたがファイルをセーブする前に，そのファイルタイプを確認しておくことは重要です」などと英語の構文そのままの日本語が出てきたら，要注意でしょう。これは，出来の悪いIMEを使っていると日本語の品質に影響する，という現象と似ています。

これを避けるためには，以下のような2段階に分けて翻訳を行うのが効率がよいようです。

1）日本語の品質は問わないから，とにかく翻訳を行う(外国語→日本語)
2）1）の結果をよりよい日本語に書き直す(日本語→日本語)

2)の作業は1)の下手な日本語に影響されるかもしれませんが，ひとりで翻訳を行うより，集中力は持続するはずです。アセンブラでなにかプログラムを作る場合，初めから自分で作らずに，コンパイラの出力結果を直しながら作るほうが集中力が持続する，のと似ているかもしれません。

1)を担当する人には，あまり品質は問われません。とにかく根気よさが必要とされます。1)と2)を分けるには，十分に時間を分ける，あるいは別な人が行う，という2種類の方法が考えられますが，よくよく考えると，1)はなにも人間が行う必要はないわけで，コンピュータでもかまわないわけです。

どんなに機械的な日本語でも，外国語よりは日本語に近いわけですから，2)の作業中の集中力も少しは持続するはずです。また，何割かの誤訳があったとしても，これが最終結果になるわけではありません。

最近の小説(特にサスペンス系)は，英語圏からの翻訳が飛び抜けてよくなっていますし，原書が出版されてからのタイムラグも短くなっています(現在，P・コーンウェルの最新作「スズメバチの巣」を読んでいますが，これはよいです。締め切りさえなければ……)。

ひょっとして，これらの成果は最近めきめき性能の上がってきた自動翻訳ソフト/翻訳支援ソフトのおかげかな？　と思っていますが，プロの翻訳者の方々いかがでしょうか？

■なぜフランス語か？

最近は，パソコンショップでも翻訳ソフト専門のコーナーが設けられていたりして，翻訳というジャンルもワープロ，表計算，データベースなどのようにパソコンの主要な用途となっているようです。ひょっとすると，CやJavaなどの開発ソフトウェアのコーナーより多くの場所を割いている店もあるかもしれません。

ところが，よく見ると，英日/日英に十数社，韓国語に1社，中国語に1社といったところで，英語に著しく偏っています。英日/日英というひとつのジャンルにこれだけのメーカがひしめきあっていれば放っておいても，この分野はどんどんよくなっていくでしょう。特に，英語のように係り受け(not only 〜 but also 〜など)が少ない言語では，最後は辞書の大きさが勝負の決め手になってきますから，このままいけば数年内には，我々アマチュアの入る余地はほとんどなくなるかもしれません。

ところが，世界第2の使用人口を誇る仏語に目を向けると，私の見つけた範囲では，英語経由の二重翻訳(仏→英→日)プログラムが1社あるだけで，あまり活発ではありません。後述するように，仏語には英語にない特徴がいくつかあって，この情報を有効に活用すると，より精度の高い翻訳ができるのですが，いったん英語に落としてしまうと十分にこの情報を活用できません。

まぁ，せっかくワールドカップもフランスで開催されたことですし，というわけで英日ではなく，仏日翻訳ソフトをぶつぶついいながら最近作っています。

■インストール方法・使い方

インストール方法は至って簡単です。ハードディスクに適当なディレクトリを作り，付録のCD-ROM DISC1中から，￥F2J￥EXEの中のファイルをすべてコピーし，install.batを実行してください。

これで，インストールは終了です。

C++のこと

COLUMN

「来来仏語」では，処理中の単語（Class名 TOKEN)は，すべて，属性情報(Member名 jpProp)を持っています。

たとえば，Franceという単語は，固有名詞(JP_PROP_UNIQUE)であると同時に，場所を表してもいます(JP_PROP_PLACE)。

最初は，これらの関連性を実現するのに，ビット演算で条件判定を行っていました。pというポインタに"France"が辞書から引っ張られてきて，pのメンバのjpPropが，その属性情報を表していた場合，

```
p->jpProp & JP_PROP_UNIQUE == 1
p->jpProp & JP_PROP_PLACE  == 1
```

という具合です。

属性の種類が32個を超えるまでは，

```
int jpProp
```

としておけば単純なビット演算でこれらの処理を実現できました。しかし，プログラムが進むにしたがって，32種類の属性では，どうにもならないことが判明しました。このとき，プログラムの大きさは，すでに8000行近くありましたし，属性判定にからむ演算は，数え切れないほどありました。このときint型変数を2つ使って，

```
int jpProp1
int jpProp2
```

情報量を増やし，

```
p->jpProp1 & JP_PROP1_PLACE
p->jpProp2 & JP_PROP2_HUMAN
```

のように，32個以上の情報をやりくりしていたのでは，これらの演算部分すべて，書き直しですし，やがて，64種類以上の情報が必要となったときには，また書き直しです。

そういえば，そんな話をどこかで聞いたよなぁ，というわけで思い出したのが，C++のClassという機能です。実は，MFC以外でC++の機能を私が使ったのは今回が初めてです。C言語の習得で頭の進化が止まってしまった方，この連載を通して遅ればせながら，いっしょにオブジェクト指向を学んでいきましょう。

クラス概念とかオブジェクトの継承とか，いきなり核心に入らずに，実利的な面から，C言語ではここが不便，C++にしたらこんなにすっきり，というように「来来仏語」で実際に感じたことを中心に説明できたらと思っています。現在，Ver. 0.31で，2万行近くありますので，そろそろ，C言語だけではやっていられなくなりそうです。適宜，C++への書き換えを行い，移行前とあとの比較などを行いたいと，思っています。

あとは，F2j.exeをクリックすると，MS-DOS窓が開き，数秒後，

仏>

と表示し，入力待ちの状態になりますので，

仏> Bonjour.

などと，適当なフランス語をキーボードから打ってください。すぐに，

日> こんにちは。

と表示されるはずです。

CD-ROMに収録した「来来仏語」は，Version 0.30で，まだまだ未完成品です。扱える構文の数も限られていますし，単語数も貧弱です。

一応の目安として，仏検3，4級くらい(学習時間200時間くらいといわれているが)のレベルの構文，単語はあると思いますが，それでも不十分な場合は名詞，形容詞，副詞は，それぞれf2j_noun.dic，f2j_adj.dic，f2j_adv.dicというファイルに格納されていますので，ユーザーが自由に拡張することが可能になっています。辞書ファイルはCSV形式でWindows95/98に付属のWord Padでも編集できるようになっていますので，積極的に追加していってください。フォーマットなどの詳細はCD-ROM中のreadme.txtの後半部分を参照してください。

前置詞，動詞はプログラム中にコーディングされていますので変更は困難です。動詞の拡張方法は次回以降に説明しますので，それまでお待ちください。前置詞はこのレベルのものはひととおり登録したつもりですが，もし，抜けている単語を見つけた場合には，編集部まで，ご一報いただけると助かります。

大学1年教養課程で仏語を勉強している友人などにテキストを借りられる場合は，その内容をいろいろ試してみましょう。数日分は昼食をごちそうになれるはずです。2年生の友人は，留年してない限り難しいかもしれません。

このとき，キーボードから入力できない文字を見かけるかもしれませんが，表1のように切り抜けてください。Windows 95/98では，コントロールパネルの国際化のところで，キーボードをフランス語対応にできますが，「来来仏語」では対応していません。

表1　英語にはない表記

仏語	成り立ち	代用文字
à	a + accent grave	@
â	a + accent circonflexe	[
ç	c + c{dille	¥
é	e + accent aigu	{
è	e + accent grave	}
ê	e + accent circonflexe	]
ë	e + tr{ma	&
î	i + accent circonflexe	^
ï	i + tr{ma	%
ò	o + accent grave	`
ù	u + accent grave	|
û	u + accent circonclexe	~

表2　「来来仏語」が使用している単語の属性（:の左は辞書ファイルに用いている記号）

行：行動
状：状態
人：人間
体：体の一部
場：場所
物：物
抽：抽象的なもの
時：時間
固：固有名詞
量：量を表す単語
衣：衣服
食：食べ物
飲：飲み物
乗：乗り物
音：音の鳴るもの，楽器など
勉：仕事というよりは勉強するもの
冠：タイトル（Monsieur, Madameなど）
材：材料
天：天候
単：単位
感：感情

■フランス語とは，どういう言語か？

基本的には，英語と同じ，

主語 + 動詞 + 目的語

が基本です。ただし，以下のような違いがあります。

1）動詞の活用

大変です。日本語より複雑です。原形，受け身形に加え，格(1人称単数/複数，2人称単数/複数，3人称単数/複数)がそれぞれ，直説法現在形，過去形，未来形，接続法現在形を持っています。命令形も3種類あります(～してください，～しましょう，～しろ)。

つまり，ひとつの動詞が，少なくとも，6×5+2+3＝35種類に活用します。この35種類の活用の種類が，80近くあります。

覚えるのは大変ですが，要はPattern Matchingですので，Pentiumの力をもってすれば，朝飯前です。

2）品詞の性

これはわりと有名ですので，皆さんもどこかで聞いたことがあると思います。フランス語には，名詞に男性名詞，女性名詞という分類があります。

ami(男性形)　　男の友達
amie(女性形)　　女の友達

のように実際の性によって，決まる場合もありますが，

cravate(女性名詞)　ネクタイ
stylo(男性名詞)　　ペン

などのように，人間の性とはまったく無関係に決まる場合がほとんどです。70%くらいの確率で，“e”で終わる名詞が女性形,それ以外が男性形です。

英語でも，shipをitではなく，sheで受ける人がいますが，おそらく，フランス語(あるいは，スペイン，イタリア語か？)の影響でしょう。

これも，外国人にとっては迷惑な話ですが，ごちゃごちゃいっていないで，PC用の辞書に入れてあります。

また，冠詞，形容詞，動詞の一部も修飾する名詞の性により形が変わりますから，この単語はどの名詞に係るのかなー，と悩んだときには，1/2の確率でこの情報が役に立ちます。また，これらの単語は，名詞が複数か単数かによっても形が違う場合がありますから，構文解析にはわりと有効です。

3）時勢

英語と同様，助動詞で作る場合と，動詞の活用で作ることができます。

eg.）came/come/will come（be going to come）
e vientiendrai/venir/venins（aller venir）

英語では，

have＋過去分詞 → 現在完了
be＋過去分詞 → 受動態

と明確なルールがありましたが，仏語では，一部の動詞で，

etre（beに相当)＋過去分詞

で，現在完了(複合過去という)と受動態の両方にとれる場合があり，プログラマの腕の見せどころとなっています。

4）目的格代名詞(補語人称代名詞)の位置

これは，英語にはありません。動詞の目的語に，代名詞を使う場合は，

主語 + 動詞 + 目的語(代名詞以外)
e 主語 + 目的語(代名詞)＋動詞

の順に入れ替えます。

eg.)Je aime Marie.
私(主) 愛する(動) マリー(目)
e Je la aime.
私(主) 彼女を(目) 愛する(動)
(本当は，縮めて，Je l'aimeという)

ですから，フランス語で，I love you.という場合は，Je t'aime.(ジュ・テーム)といいます(te：あなたを)。

5）やたらと単語を増やさない

たとえば，英語では風呂場(Bathroom)，警官(Policeman)と1語ですが，仏語では，

salle de bain　(room of bath)
風呂の部屋→　風呂場
agent de police　(agent of police)
警察の係員→　警官

というようになるべく既存の単語の組み合わせで表そうとします。少ない単語で英語より多くのことを表現できるわけです。

翻訳プログラムを作成する場合，単語数をいかに稼ぐかが鍵になってきますが，仏語の場合，ひとつの単語の使用範囲が英語に比べて広いので，有利です。

■単語の属性

市販の英日翻訳ソフトを使っていて，あまりにも機械的な翻訳結果に不満な場合があります。たとえば，

I have a car.　　私は車を持っている。

という翻訳結果に対して，

I have a brother.　　私は弟を持っている。

ではなく，

I have a brother.　　私には弟がいる。

としてもらいたいものです。

「来来仏語」では，これを表2にあるような，属性情報を単語辞書に加えることによって，実現しています。

例）J'ai un voiture.　　私は車を持っている。

J'ai un frere.　　私には兄弟がいる。

（実際には，表1のような変換を行ってから，入力してください）

ある本によると，自然言語のすべての単語は，7種類，あるいは11種類の属性に分類できるという主張があるそうですが，「来来仏語」では，30種類以上の分類を使わなくてはなりませんでした（含，内部処理専用。今後，さらに増える予定）。

場合によっては，

I take a pen.　　私はペンを取る。

I take a train.　　私は列車に乗る。

I take a picture.　　私は写真を撮る。

という細かい区別もあります（ちなみに，フランス語では，上から，

Je prends un stylo.

Je prends un train.

Je prends une photographie.

です。付録CD-ROMに入っている「来来仏語」で，試してみてね）。

本によっては，素性とか概念とかいう場合がありますが，要は，その単語がどういう種類かの情報によって，よりよい翻訳を目指そうね，ということです。

■英日翻訳したい人へ

今回のプログラムは，仏日ということで，まとめてありますが，基本的な構造は，ヨーロッパ系の言語に使えるはずです。辞書を入れ替え，目的格代名詞の処理ルーチン（ProcessObjective Pronoun）を無効にすれば，それだけで，かなりの英文が処理できると思います（英語，ロマンス系言語以外，あまり自信がないが）。

英日翻訳というのは，辞書の勝負で，プログラマから見るとあまり面白味がないと思うのですが，市販の翻訳ソフトに我慢できなかったり，一矢報いてやろうと考えている方は，この記事，付録CD-ROM中のプログラムを参考に各自がんばってください。がんばった方は，なぜか，知らぬ間に編集スタッフになっているというOh!Xの伝統も忘れないでください。

■謝辞

自動翻訳システムを考えた場合，プログラム本体と同様に，重要な役割を果たすのが，単語辞書の存在です。「来来仏語」では，江崎リエ子氏がNIFTY SERVEのFLR（外国語フォーラム·ロマンス語派分館）で主催されている読書会の記録より，単語調査部分を抽出し，使用させていただきました。長きにわたり読書会を主催されてきた氏の情熱と労，そして，データ使用許可をくださった寛容に心より感謝いたします。

■下回号見！ さようなら

どういうわけか，最近，中国語（北京語）を習っています。英会話学校の併設コースなので，お約束の「これは，ペンですか，それとも机ですか？」（這是鋼筆，環是卓子？）とかやっています。テキストも英会話クラスと共有なので，「マリー瑪雨小姐はピータ比太先生より背が高いですか？」となります（字体の関係でいちばん近い字を当てていますが，細かい違いは気にしないでね）。

「（マライアキャリー）瑪麗亜凱莉の歌は好きですか？」

は，発音も近く，なんとなくわかりますが，やはり生徒は日本人ですから，日本の話題が入ってくるわけで，「南十字星合唱隊的怎磨祥？」といわれると，ハタと困るわけです（ちなみに，今年で活動20周年を迎えたバンドです）。

慣れれば，「可愛的エリー○雨真好听」と答えられますが，必ずしも1対1で対応していないのがミソで，同系の「管」だけではなんのことかわかりませんが，内容を説明する語を加えて「夏の管」で，チューブだそうです。

こんなプログラムを作っている関係上，授業中は，どうしても頭の中はToken Treeが，ニョキニョキ育っています。中国語とは恐ろしい言語で，かなりの頻度で主語は省略するわ/かといって主語が2つあったりするわ/接続詞はないわ/関係代名詞はほとんどないわ，で基本的に思いついたことを口から順に出していくということになっているようです。

私としては，自動翻訳に挑戦してみたい言語ではあるのですが，いまのところ，文字コードの問題があり，躊躇しています。中国語は日本語のシフトJISと同じようなやり方で（台湾/シンガポール方式をBG Code，中華人民共和国方式をBIG5 Codeと呼ぶらしい），英数字しかないASCII Codeの拡張を行っているのですが，当然，日本のシフトJISと互換性はありません。拡張方式は同じですので，フォントをちゃんと切り替えてやれば，文章中の表示は問題ないのですが，C++のソースコード中で，フォント表示によって，見えたり見えなかったりするデータがあるのは，とっても不便です。そこで，これらの問題を解決するコード体系として，UNICODEが期待されるのですが，Windows 98での対応はどうなるのでしょう？　かなり好意的な評価も聞こえてきますが……。Windows 98（とVisual C++かな）でUNICODEが真剣にサポートされていたら，中国語もやりたいなぁ，と考えています。

本バージョンの制限

COLUMN

本文で，偉そうなことを書いたので，過度の期待をなさる方がいないよう，あらかじめいいわけさせてもらいます。

まず，今回のバージョンの「来来仏語」は，Ver 0.30であり，完成品ではありません。単語数が少ないのはともかくとして（約7000語），

熟語を使用した文

<<>>などの特殊文字つきの文

倒置が行われた文

単語の一部が省略された文

などは，いまのところ対応していません。また，基本構文を処理するのに最適なプログラム，データ構造を模索している段階で，無理やり上記の処理を加えても，基本を変更したら，またやり直さなければならないからです。たとえば，ひと口に熟語といっても，

形容詞＋名詞

動詞＋目的語

動詞＋（形容詞＋名詞）

などがあり，それぞれプログラム内部では処理する個所が異なります。

そのほかにも，

コンマ"，"の処理

が思っていたより複雑で苦労しています。

これらは仏語に限らず他言語の翻訳にも役立つはずですので，ある程度，考えがまとまった時点で他言語の例をを交え，説明したいと思います。

シミュレータ指向でゲームを作る

丹　明彦/Tan Akihiko

今後進んでくるゲームのオブジェクト化にともなう変化のうち，もっとも重要なものが物理モデルでのシミュレートだ。物体の動作をすべて指定することからシミュレートによる動作に切り換えることによって，表現的にも工程的にも新しい世界が見えてくるはずだ。ゲーム性やゲームの構造自体にも当然影響を与えるものになるだろう。今後のキーワードのひとつになってくる概念を基本の基本から押さえていってみよう。

画面の中に世界を感じるゲームが好きだ。ゲーム世界を司る法則を一貫性をもって作り上げ，個々の存在を例外なくその法則に従わせる。運動や物事の関係性に対して誠実さを通し，御都合主義に支配されていない。そういうゲームの作り方が好きだし，自分でもそういう作り方をしたい。

最近では，傑作トラップアクション「影牢～刻命館 真章～」(PlayStation)が好みだ。最初は獲物を罠にかけるのが楽しい。そのうち個々のトラップの挙動や複数のトラップの関係性がしっかりしていることに気づく。創意工夫によってトラップの連携に無数のバリエーションを編み出すのが楽しくなる。実に素晴らしい。

本稿では，こうした"ワールドシミュレータ"を目指すための方法論のひとつとして，主に力学法則をベースとしたモデル化への導入を行ってみたい。

■世界を作るということ

冒頭でも述べたように，ゲーム設計においては，計算機の中に世界を構築してシミュレートすることを指向したいと考えている。

ただし，シミュレーションとはいっても必ずしも現実の物理法則に従うことはないとも考えている。世界の設計はゲームの性格によって「現実そのものの模倣」から「架空の世界」まで柔軟に変化させてよいし，そうしなくてはならない。それは，物理定数を調整して遊びやすくすることだったり，大胆に架空の世界をこしらえて法則を体系だてることだったりする。

いずれにせよ，その世界の中に存在する物体が存在感を持っているという点は譲れない。物体どうしが近付きまたは接触すればなんらかの相互作用が生じるべきだし，物体の自発的な動きにも説得力がなくてはならない。

それにはきちんとした根拠に裏づけられたふるまい(挙動)をすることが必要となる。モデル化とは，つまりこの世界を設計し挙動を与えることといえる。

■モデル化

筆者が挙動モデルを設計するにあたっては，概念的だが，

「結果を直接プログラムせず，仕組みをプログラムする」

を基本方針としている。

対象となる事象をよく観察し，そのメカニズムを見切る。そしてそれを徹底的にプログラミングする。

たとえば，自動車の動きなら，アクセルを踏めば車が前に進み，ステアリングを切れば車が左右に回転する，と操作を直接結果に反映させるよりも，エンジンと駆動系とサスペンションとタイヤのメカニズムをプログラムとして作り込み，結果として自動車らしい動きになるようにしたい。

たとえば，人間の動きなら，四肢の動き(3D的にいえば関節モデルの座標と回転角)をトレースするよりも，骨格と筋肉を持ったモデルの挙動を関節まわりのトルクで記述するというアプローチを採りたい。外見も，皮膚を直接変形させるのではなく，筋肉や内臓組織の運動によって表皮の動きを決定するように作りたい。

これにより，人為的に結果だけをプログラムしたのではなかなか実現できない挙動のバリエーションが自然に出てくる。

この方法論はかなりの高確率でギャンブルとなる。なぜなら，ものごとのメカニズムをきちんと見切れなかったり，立てたモデルに要素が足りなかったりすると，挙動のバリエーションが出るどころか，基本的な動きさえも変になってしまうからである。納期の厳しい商用ソフトでは，これはきわめて大きなリスクとなりうる。

しかし，これは冒す価値のあるリスクなのだ。もし革新的なプロジェクトを成功させたければ，このリスクは避けて通れない。プロジェクトの主導者には，このリスクを背負えるだけの度量が必要なのだといい切ってしまおう。

■力学モデルのシミュレーション

ニュートン力学の範囲内で起こるできごとをモ

実数の表現――浮動小数点演算と固定小数点演算

COLUMN

本稿で取り扱っている力学シミュレーションなどのような自然科学系シミュレーションを行うには，なんらかのかたちで実数を計算しなくてはならない機会が多くなる。

基本的に整数のみを扱うデジタル計算機で実数を処理するためには，整数データをなんらかの手段を用いて実数として扱うルールを確立し，事実上の実数計算を行うというのが妥当な方法論といえる。そのなかで私がメジャーだと思っている方式が浮動小数点演算と固定小数点演算である。

浮動小数点演算は一定長の整数データをビットフィールド表現で符号と指数部と仮数部に分け，仮数部の値を指数表現によってスケーリングすることで小さな値から大きな値までを表現する方式である。固定小数点演算は整数データの下から何ビットかを小数部，残りを整数部として扱うと取り決め，ある一定範囲の値を表現する方式である。

浮動小数点演算方式は，同じビット長ならば，固定小数点演算方式よりも値のダイナミックレンジを幅広く取ることができる。指数部にビットフィールドを分け与えているため有効精度はやや落ちるのだが，どのスケールの値でも安定して精度を確保できる。しかし値のハンドリングはビットフィールドの演算を伴うためそれなりに複雑で，システムが専用のハードウェア(FPU)を持たない場合はまず実用にならない。

固定小数点演算方式は，整数データの全ビットを使えるので，範囲が適正であれば浮動小数点演算方式よりも精度が高く取れるが，範囲が適正でないと精度が悪化したり，最悪の場合は値が扱える範囲を越えてしまい，値を表現することそのものが不可能になる。しかし値のハンドリングは比較的楽で，CPUが整数演算しかサポートしていていなくても比較的実用になる負荷で計算できる。

どちらが優れているかは，そのプログラムが動作するハードウェアに依存する。FPUを搭載していないシステムであれば問答無用で固定小数点演算を用いなくてはならないが，十分に速いFPUを搭載しているシステムであれば，総合的に見て浮動小数点演算方式を採用すべきだと思う。

デル化するなら，
・直線運動
・回転運動
の2つのシミュレーションを道具として使いこなせればおおむね問題ないので，これらの解説を行いたい。微分/積分の知識をもとにしているが，数式そのものが苦手というのでない限り理解できるように配慮したつもりだ。

先ほど述べた"事象のメカニズムを見切る"とは，力学シミュレーションの場合，運動を決定する要素を上の2つに分解する作業にほかならない。どうやって分解するか(どうモデル化するか)について具体的な方法を私は知らない。いつも問題に対面してからその場で考え，その場で解決しているからだ。またそれだからこそモデル化は非常に面白い知的作業ともいえる。

シンプルで素性のよい，できるだけ少数の式に分解するのが理想だが，それができるようになるには訓練を積むしかない。才能ということはないと思う。

難しい問題に対して鮮やかなモデルが閃くと至福の瞬間を得られる。ぜひ味わってもらいたい。

テレビのフレームレート

COLUMN

日本のテレビ信号（NTSC）は毎秒30フレームのインタレース走査であり，1フレームは2フィールドで構成されている。フィールドとは画面を構成する走査線を奇数本目と偶数本目に分けた呼び名。インタレース方式はあるサイクルで奇数本目をリフレッシュし，その次のサイクルで偶数本目をリフレッシュするというように交互にリフレッシュする方式。したがって，NTSCの毎秒30フレームインタレースはいい換えれば毎秒60フィールドである。

家庭用ゲーム機では各フィールドに違う映像を送り込むことで，60fpsを事実上実現している。映像信号の規格に厳しく則れば，NTSCはあくまで30fpsであり60fpsの表示という表現は誤りということになるが，私はこれに「論理フレームレート」というべき概念を導入すればよいと考えている。

論理フレームレートは，アプリケーションで表示を変化させる頻度と定義すればよい。ポリゴン指向のハードウェアであれば，ダブルバッファのスワップバッファを行う頻度が論理フレームレートとなる。

物理フレームレートは常に30fpsで固定，しかし論理フレームレートはアプリケーションにより可変というわけである。処理の重たいアプリケーションでは20fpsだったり15fpsだったりするが，これも論理フレームレートで片がつく。

ところで，ゲーム機の60fpsというアプリケーションを見ていると，インタレース表示になっていないことがあるのだが，あれはNTSC規格に沿っているのかどうかちょっと自信がない。もしそういう動作モードがあるとすれば，物理的にも60fpsということになるのであろうか。

図1　速度が一定な場合の運動シミュレーション

■計算機におけるリアルタイムシミュレーションとは？

本稿にて指向しているゲームはリアルタイムシミュレーションである。リアルタイムシミュレーションにおいては，瞬間瞬間のものごとの状態を計算し，結果を表示する。シミュレーション世界での時間の流れは現実世界と同じである。

ただ，リアルタイムシミュレーションといっても，完全に現実と同じ時間の流れをしているわけではない。というのは，コンピュータ(正確には現在のPCや家庭用テレビゲーム機)は毎秒数十回というサイクルで画面をリフレッシュするという表示方式を用いており，これが動きを表現できる細かさの限界となる。

この画面リフレッシュのサイクル数をフレームレートと呼ぶ。ゲーム雑誌を読むとたまに毎秒30フレームとか60フレームという用語を目にすることがあるが，これがフレームレートである。単位はfps(frame per sec，毎秒～フレームを意味する)。

そして，シミュレーションのサイクルも基本的にはこのフレームレートに同期させるのが，シミュレーション結果を過不足なく表示できるという点で合理的といえる。30fpsであれば毎秒30回のシミュレーションを行って表示に反映させる。そしてシミュレーション時間は1サイクルにつき1/30秒ずつ進ませるということになる。

このシミュレーションの1フレームあたりの経過時間をΔtとする。Δtはリアルタイムシミュレーションにて最初から最後までつきまとう大事な量である。

なお，PCアプリケーションにおいてはCPUパワーやビデオカードのパフォーマンスがハードウェアでまちまちなため，構造上フレームレートを固定できない。また，PCでもゲーム機でも，3Dアプリケーションに顕著な処理落ち･描画落ちのためにフレームレートが安定しないことがある。こうした場合のシミュレーション時間の決め方にはそれなりの戦略が必要なのであるが，今回はそこには踏み込まないことにする。

■直線運動 その1：等速運動

たいへん前置きが長くなってしまったが，いまやΔtを手にしたので力学シミュレーションの実際に進むことができる。

先ほども述べたが，リアルタイムシミュレーションとは瞬間瞬間のものごとの状態を計算する技術である。力学シミュレーションの場合，毎フレームごとの物体の位置と姿勢を計算する処理を行う。

まず力学の分野でもっとも簡単な運動，等速直線運動のシミュレーションを行ってみよう。速度がv(m/s)で一定である物体の運動である。日常的には速度の単位にはkm/hを用いているが，力学シミュレーションではm/sのほうが扱いやすい。換算は簡単で，1(m/s)=3.6(km/h)の関係があるから時速の値を与えられたらそれを3.6で割れば秒速の値になる。

ここで基本に立ち返る。速度とはなにか。それは，(物体が)単位時間に移動する距離である。数式で書けば，

$$v = \Delta x/\Delta t \quad \cdots\cdots ①$$

である。ならばある速度で移動する物体が1フレームに移動する距離Δxはどうやって求めるのか。

上の式を変形してΔxについて解いた式，

$$\Delta x = v \cdot \Delta t \quad \cdots\cdots ②$$

を計算すればよい。したがって，物体の座標値xにここで求めたΔxを加算するという操作を毎フ

レーム行えばいいということになる(図1)。

■直線運動 その2：加速度運動

次に，簡単だがもっとも重要な力学の式，運動方程式を用いた加速度運動のシミュレーションを行う。

加速度運動は，時間につれて速度が変化する運動のことである。どのくらい変化するかを表す値を加速度という。物体の運動する速度を変化させるのは力である。F(N)の力が質量m(kg)の物体に働くとき生じる加速度a(m/s₂)の関係は次の有名な運動方程式で表される。

$F = m \cdot a$ …… ③

力の単位「N」はニュートンと読む。質量1kgの物体に1m/sの加速度を生じさせる力を1Nと定義している。

加速度は単位時間に速度の変化する量のことであるから，速度と時間で表現すると，上の①とよく似た式，

$a = \Delta v / \Delta t$ …… ④

が得られる。ここまでくればもはやシミュレーションに持ち込む方法は明解であろう。運動方程式③を変形した式，

$a = F/m$ …… ⑤

と④を同時に解いた式，

$\Delta v = (F/m)\Delta t$ …… ⑥

に従って速度vを毎フレーム変化させていけばよいのである(図2)。

こうして得られた毎フレーム変化する速度をそのまま前項の等速直線運動シミュレーションに適用すれば，一般的な直線運動のシミュレーションとなる(図3)。

■回転運動 その1：トルク

さてここからは回転運動の解説となる。次々に新しい(懐かしい？)概念が出てくるが，やっていることはそれほど変わらない。直線運動と回転運動には類似した関係があり，運動を表す各種の物理量と方程式を対応づけることができる(図4)。回転運動の法則のシミュレーションへの適用もまた直線運動と同様に行うことができるのである。

まず，「力」の回転運動版ともいえる「トルク」である。トルクは物体に回転を生じさせる"力のようなもの"である。図5のようにボルトをスパナで締めつけるとき，ボルトの軸にかかるのがトルクである。

トルクの大きさは支点から力点までの距離lに，力点に与えた力Fを掛けた値である。仮にスパナを短く持って距離を半分にし，加える力を2倍にすれば，ボルトにかかるトルクの値は同じになる(図6)。これがいわゆる，てこの原理である。

一般に回転のシミュレーションにおいては，力点に加わる力がすべて回転に使われるとは限らない。力点に加えられた力のうちトルクとして一部の成分だけが有効になる場合がある。具体的には，支点と力点を結ぶ線に垂直な力の成分がトルクとして有効になる(図7)。

日常生活で，荷物の真ん中をまっすぐ押しても荷物は回ったりしないが，端を押すと荷物は回転してしまうというという経験をお持ちなら，このことが納得できることと思う。なお，力のベクトルをある成分とそれ以外の成分に分解する方法を図8に示した。

回転運動において，軸が固定されているかどうかを見落としてはならない。図7の例は物体が軸に固定されていて軸を中心にして自由に回転でき

図2　運動方式を用いた，加速度による速度変化の計算

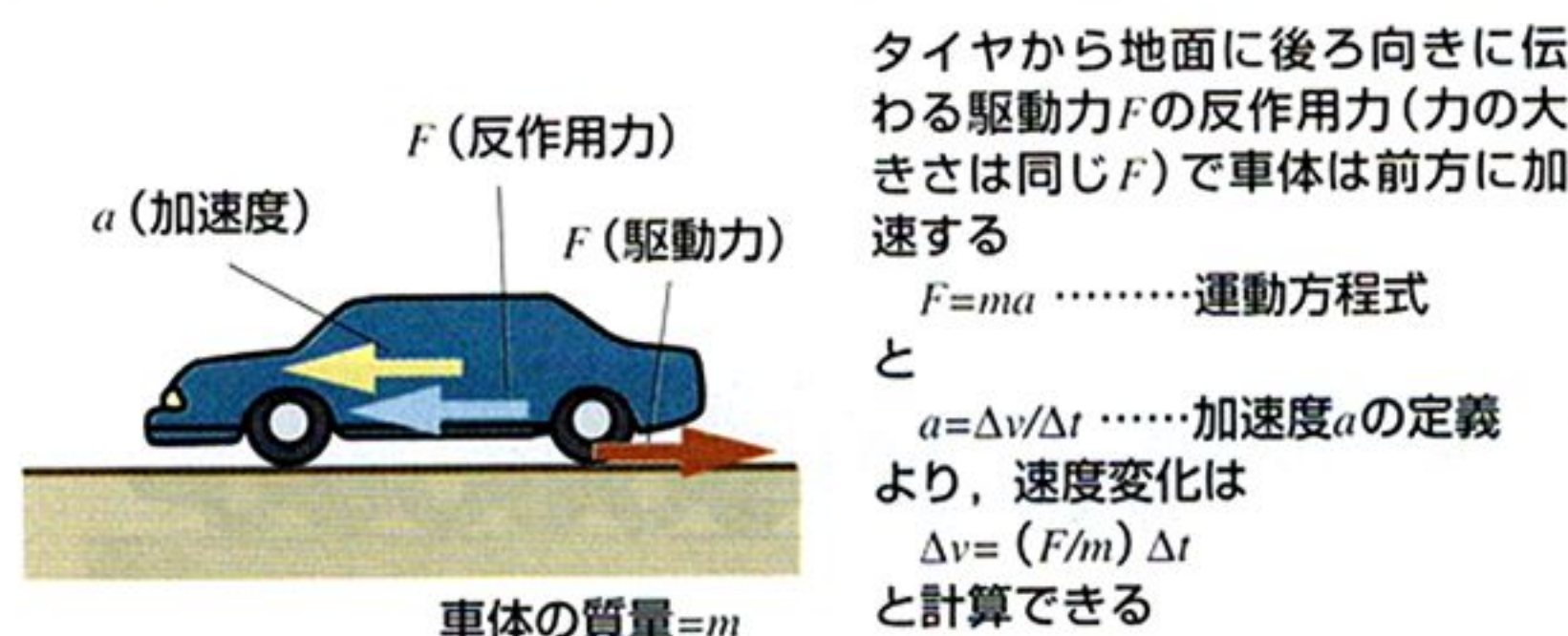



図3　速度変化を加味した運動シミュレーション

図5　トルク

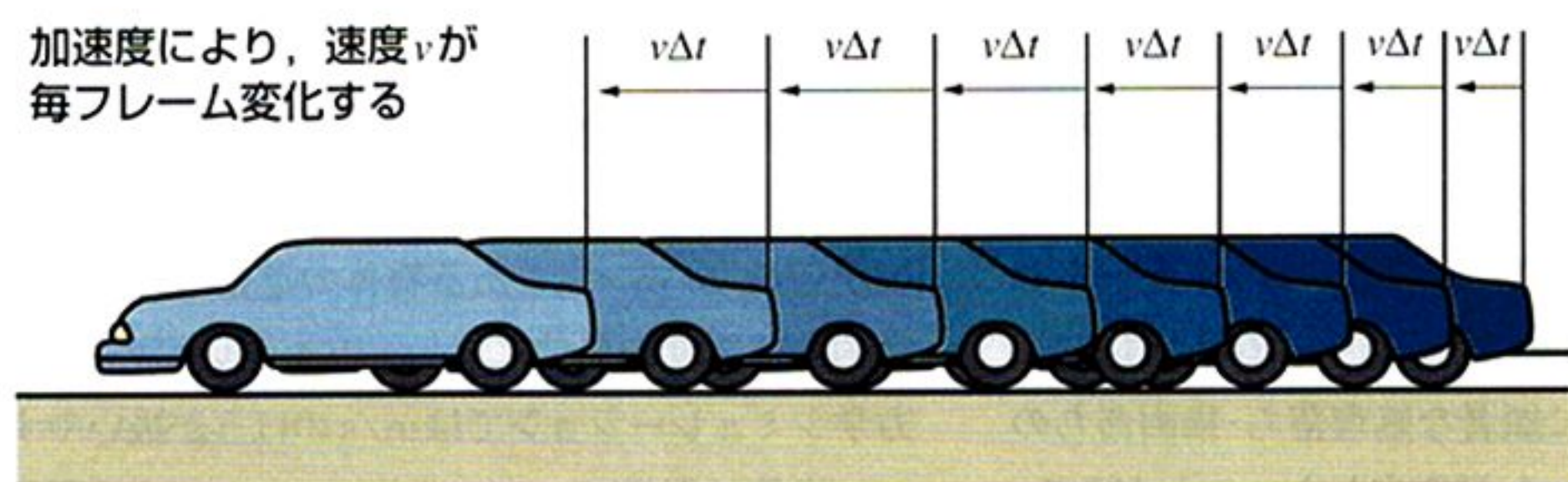



図4　直線運動と回転運動のシミュレーションに用いる各種物理量と方程式の対応

直線運動	質量	力	位置	速度	加速度	運動方程式
(記号)	m	F	x	$v=\Delta x/\Delta t$	$a=\Delta v/\Delta t$	$F=ma=m\cdot(\Delta v/\Delta t)$
(単位)	kg	N	m	m/s	m/s²	–

回転運動	慣性モーメント	トルク	角度	角速度	角加速度	運動方程式
(記号)	I	T	θ	$\omega=\Delta\theta/\Delta t$	$\Delta\omega/\Delta t$	$T=I\cdot(\Delta\omega/\Delta t)$
(単位)	kg・m²	N・m	rad	rad/s	rad/s²	–

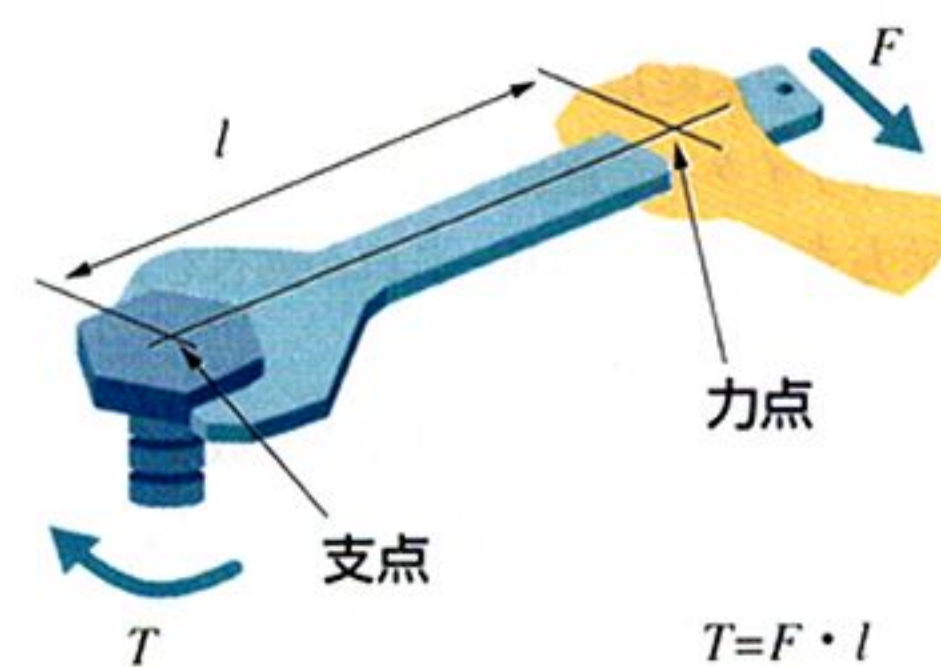
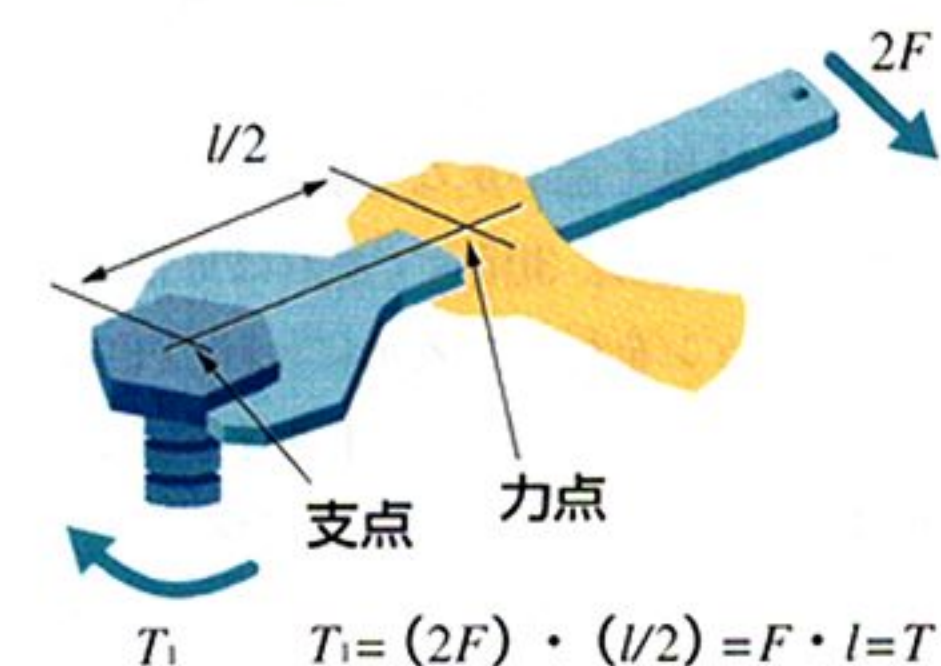



図6　てこの原理

る。この場合，力のトルクとして有効な成分(F2)以外の成分(F1)は，軸に向かって押しつける力となる。軸が丈夫なら，この力はなんの影響も及ぼさない。

しかし図9のように物体が軸に固定されていない物体だと状況は変わる。この場合，まずトルクの支点はその物体の重心(質量の中心)になる。次は同様に支点と力点を結ぶ線と垂直な成分(F2)がトルクとなるのだが，残った成分(F1)はその物体を押し出す力，つまりその物体に加速度を生じさせる力になるのである。

以上をまとめると，物体に与えられる外力は，回転運動を起こす力と直線運動を起こす力に分解してシミュレーションを行わなければならない。

■回転運動 その2：角速度

名前からも容易に推測できるように，「速度」の回転運動版は「角速度」ω，姿勢角度θが単位時間に変化する量である。定義によって式に書けば，

$$\omega = \Delta\theta/\Delta t \quad \cdots\cdots ⑦$$

ということになる。シミュレーションも直線運動と同様に，

$$\Delta\theta = \omega \cdot \Delta t \quad \cdots\cdots ⑧$$

の形式に変換して行うことができる。毎フレームこのΔθを姿勢角度θに加算すればよい(図10)。

■回転運動 その3：角加速度と回転運動の方程式

これまた名前から推測できるが，「加速度」の回転運動版が「角加速度」Δω/Δtである。角加速度を生じさせるのはトルクであり，その関係は次のようになっている。

トルクTと角加速度Δω/Δtと慣性モーメントI(これは次項で解説する)の間には，力と加速度と質量の間に成立するのとよく似た関係

$$T = I \cdot (\Delta\omega/\Delta t) \quad \cdots\cdots ⑨$$

がある。この回転運動の方程式から，角速度の変化を，

$$\Delta\omega = (T/I)\Delta t \quad \cdots\cdots ⑩$$

によって求めることができる。このΔωを毎フレーム角速度ωに加算すれば，角加速度を加味した回転運動のシミュレーションが成立する(図11)。

■回転運動 その4：慣性モーメント

慣性モーメントは「質量」の回転運動版である。今回もっとも馴染みのない概念かもしれない。言葉で説明すれば，

ある物体のある軸に対する回転しやすさを決める量

となる。手にものを持って，手首をひねって回転させようとしたとき，より力が必要であれば慣性モーメントが大きいという。

慣性モーメントを計算するのはかなり大変だ。一応，図12に定義を示す。現実の問題でこれを直接利用することはあまりないだろう。一見単純そうな積分の式だが，少し形状が複雑になっただけでたちまち計算不可能になるのである。

図7　軸に固定されている物体に働く力とトルク

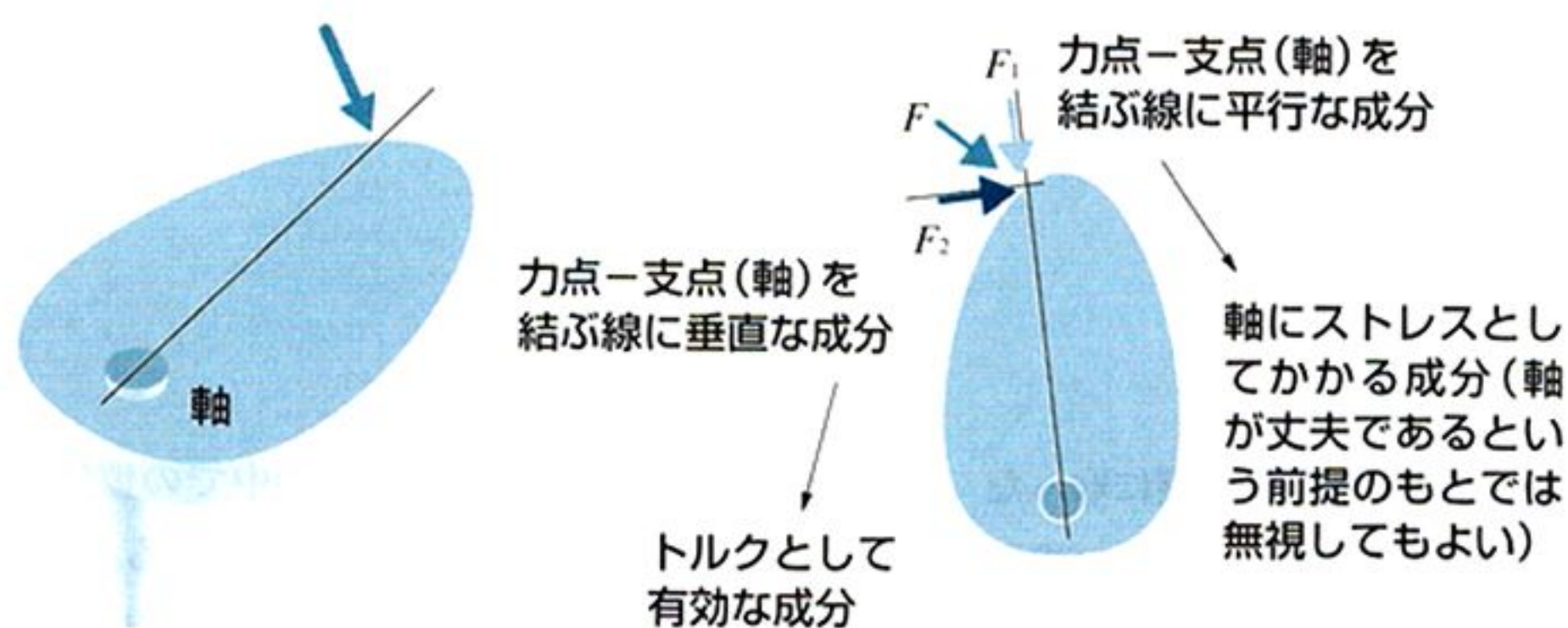



図8　力の成分の分解

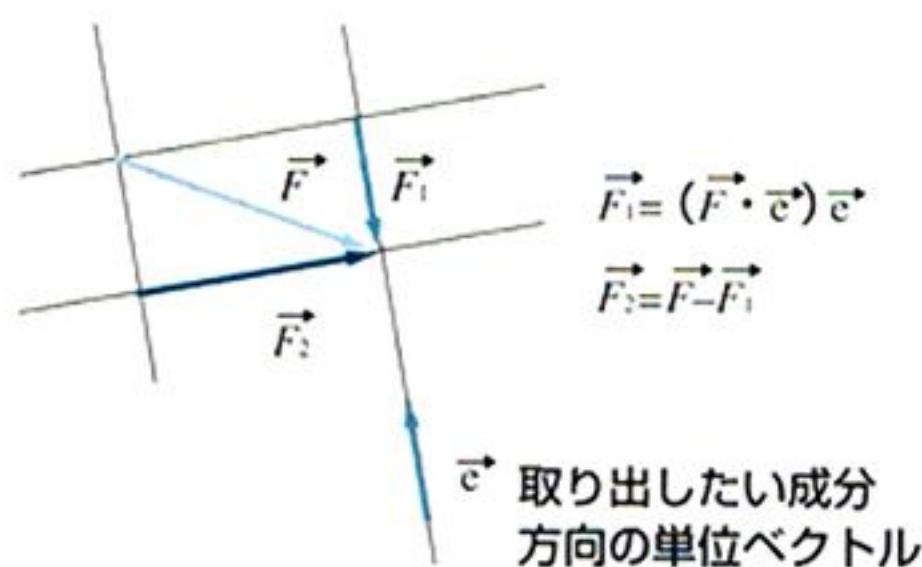



図9　固定されていない物体に働く力とトルク

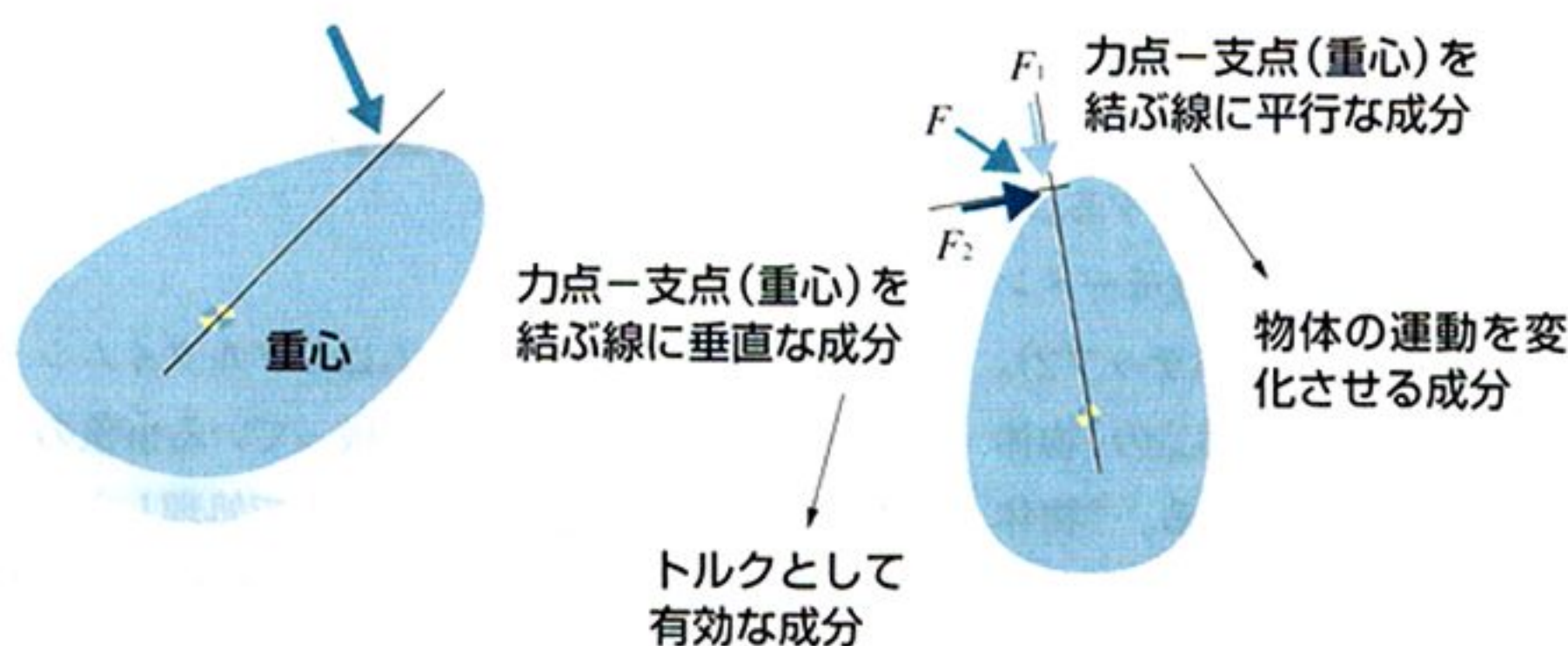



図10　角速度一定の回転シミュレーション

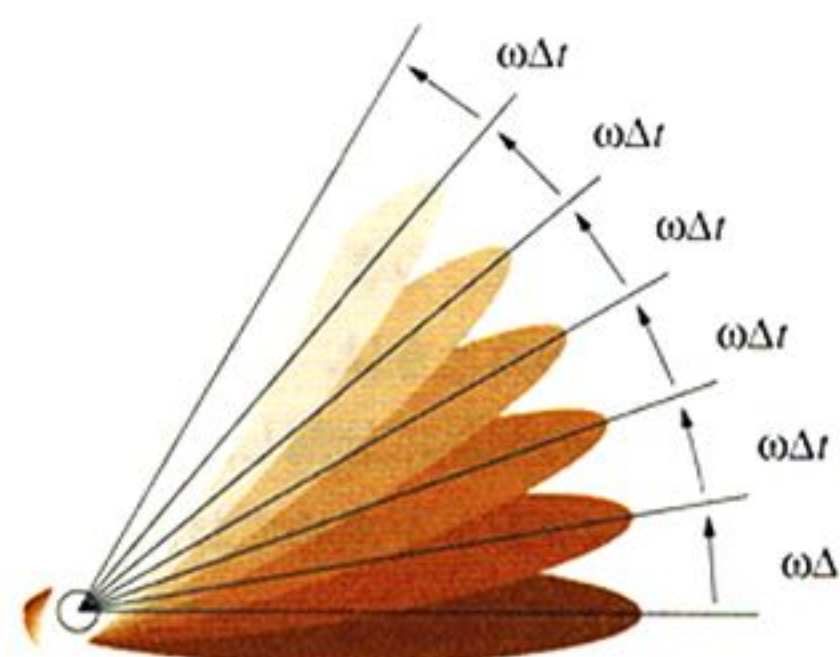



図11　角加速度を加味した回転運動のシミュレーション

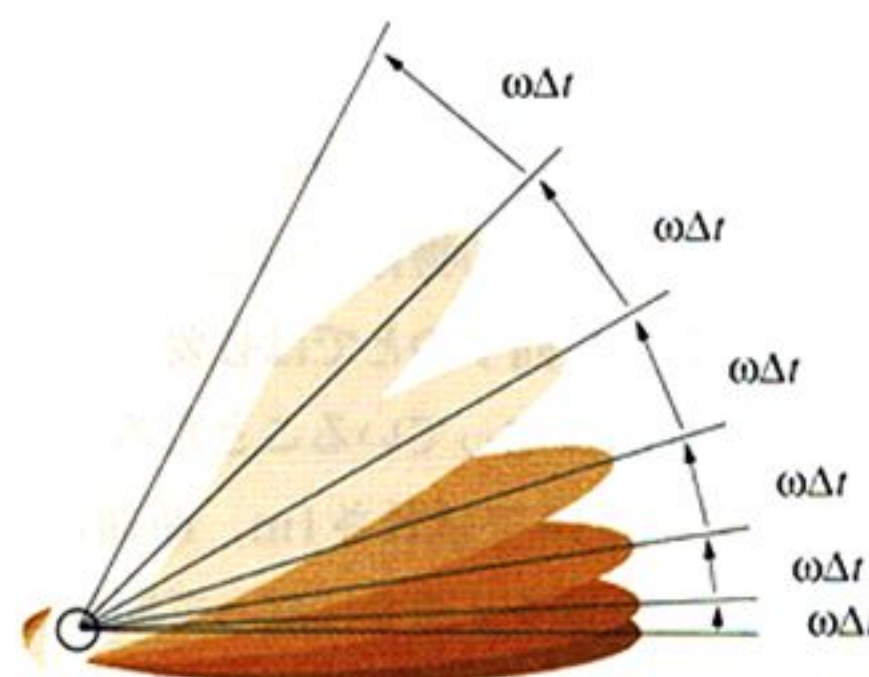



図12　慣性モーメントの計算

ある物体を構成する微小領域のそれぞれについて，軸からの距離rの2乗を質量にかけた値を求め，それを物体全体について合計した値がその物体の軸に関する慣性モーメントになる

これは次のように積分の形で計算できる

$$I=\iiint pr^2 dxdydz$$

pは密度（単位体積あたりの質量）で，dxdydzはその微小領域の体積であるから，p dxdydzはその微小領域の質量となる

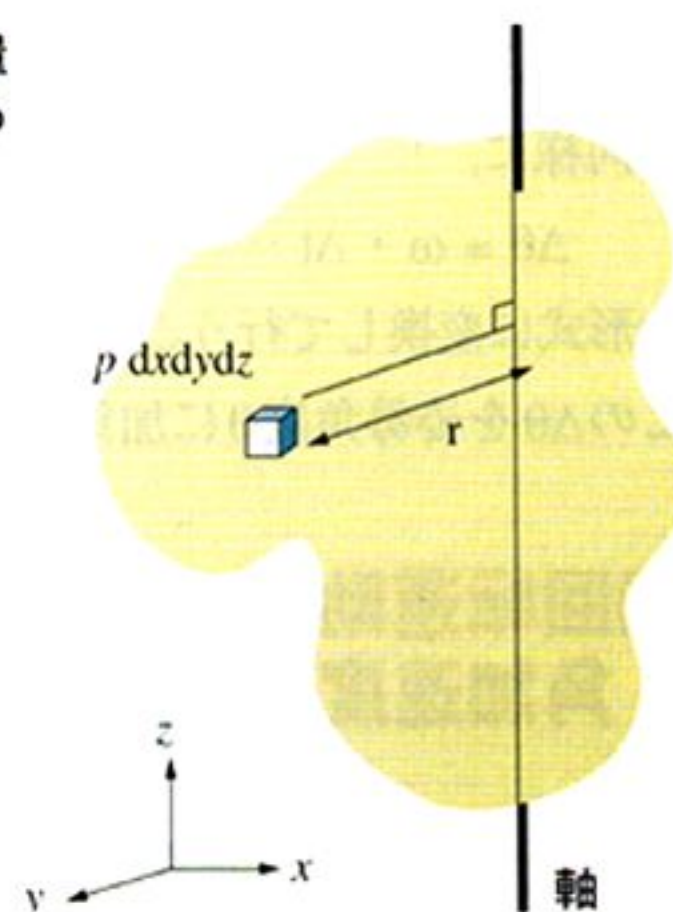
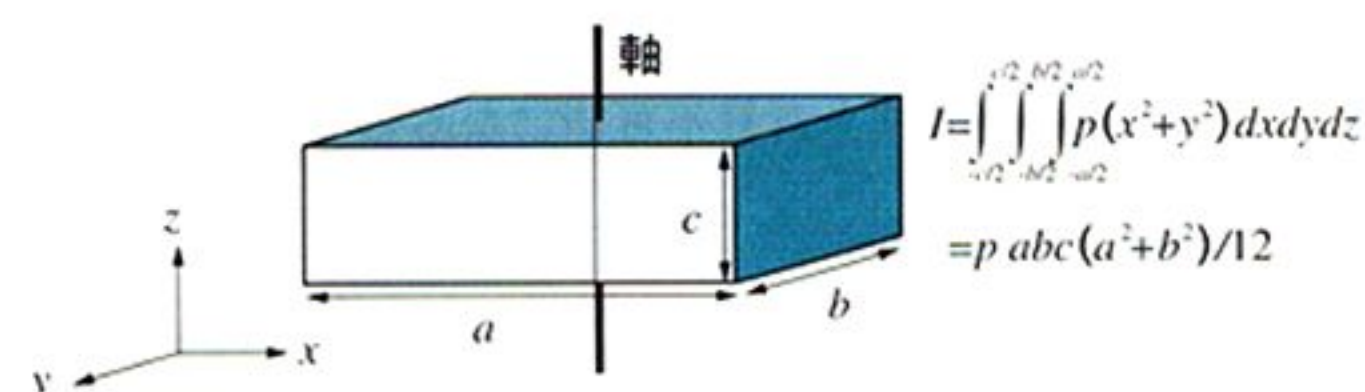



図13　慣性モーメントの計算例

一様な材料で作られた（＝密度が一定な）直方体の慣性モーメントを求める
軸は直方体の中央を，側面に平行に通っている

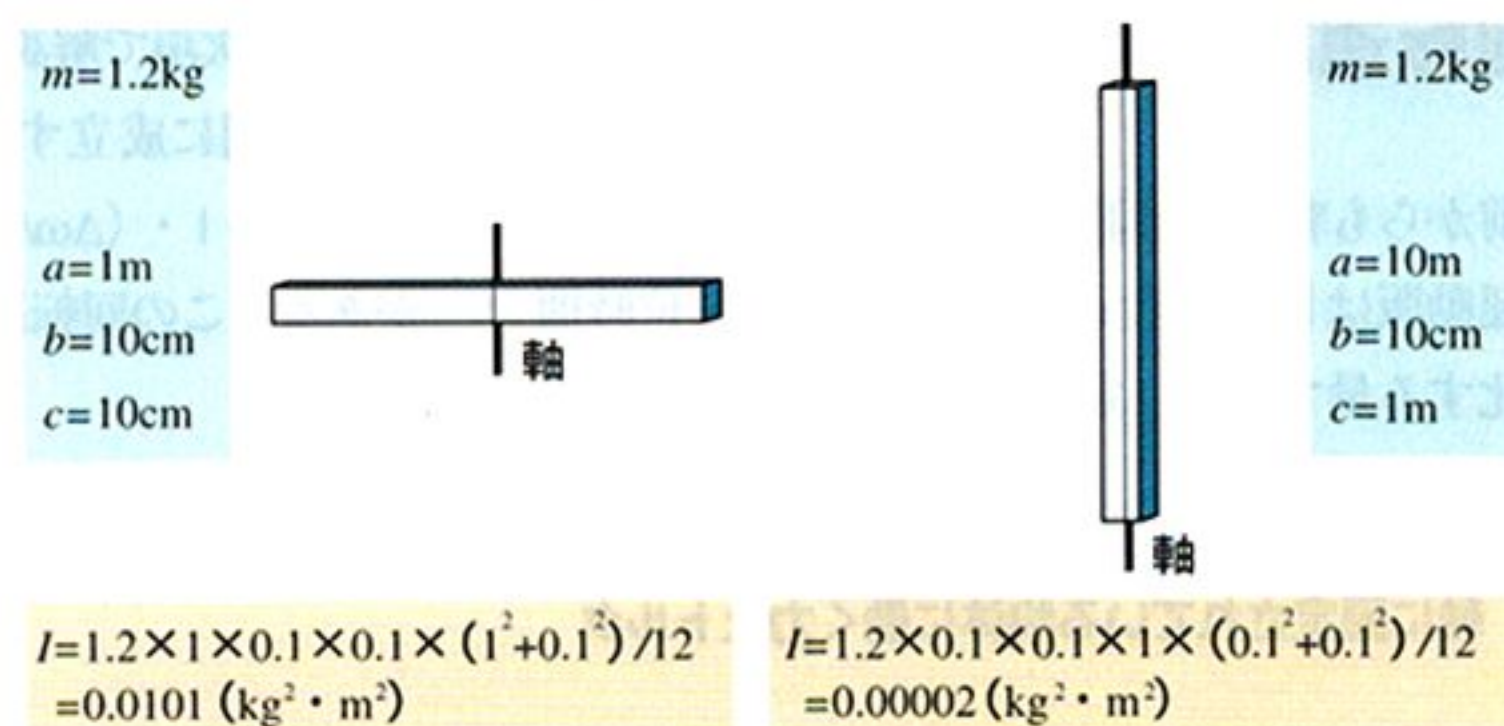
$$I=\int_{-c/2}^{c/2}\int_{-b/2}^{b/2}\int_{-a/2}^{a/2} p(x^2+y^2)dxdydz$$
$$=p\,abc(a^2+b^2)/12$$

直方体全体の質量m＝（密度）×（体積）＝$p\ abc$なので，直方体の慣性モーメントは質量を用いて，

$$I=m(a^2+b^2)/12$$

と書くこともでき，このほうが公式としては扱いやすい

図14　慣性モーメントは同じ物体でも回転軸の取り方によって異なる

I=1.2×1×0.1×0.1×$(1^2+0.1^2)$/12
=0.0101 (kg^2・m^2)

I=1.2×0.1×0.1×1×$(0.1^2+0.1^2)$/12
=0.00002 (kg^2・m^2)

横にして回すよりも縦にして回す方が回しやすい

しかし解決法はあって，慣性モーメントは合成できるのである（次項）。つまり，複雑な物体の慣性モーメントを求めるときは，まずその物体をより単純ないくつかの部品に分割し，次にそれぞれの質量と慣性モーメントを求めておき，最後に慣性モーメントを合成すればよいのである。

ということは，簡単な形状の物体の慣性モーメントを公式として持っておけばほとんどの問題を解決できることになる。

図13で，均質な材料で作られた直方体の慣性モーメントを求めている。この程度の形状なら，積分も楽に計算することができる。それも面倒なら，結果だけを公式として覚えておけばよい。

この公式を用いて，慣性モーメントの重要な性質，

　慣性モーメントは同じ物体でも回転軸の取り方によって異なる

を示しておく。先ほど手にものを持って回す話を書いたが，これが棒状の物体だった場合，横にして回すのと縦にして回すのとでは必要な力が違うというのは経験的に知っていることだろう。それを数式で示す。図14では長さ1m，断面が10cm角の角材を，回転軸を2通りに取って回転させようとした場合の慣性モーメントを求めている。結果もその経験則を裏づけている。軸を縦に取ったほうが慣性モーメントの値は圧倒的に小さいのである。

■回転運動 その5：慣性モーメントの合成

複雑な物体を簡単な部品に分割し，それぞれの質量と慣性モーメントがわかっている場合には物体全体の慣性モーメントは合成することができる（図15）。

慣性モーメントを合成する前に，回転中心を決定しなくてはならない（図15のステップ1）。物体が軸に固定されている場合は簡単で，その軸が回転中心となる。そうでない場合は物体の重心が回転中心となるのだが，重心が不明である場合は，重心を合成によって求める。具体的には，各部品の重心位置を質量によって重みづけ平均する。

回転中心が決まれば，本題である慣性モーメントの合成を行うことができる（図15のステップ2）。

物体全体の慣性モーメントは，各部品の“物体の中での”慣性モーメントの合計である。“物体の中での”とわざわざ強調したのは，慣性モーメントは軸の取り方で異なるためである。部品単体の慣性モーメントはその物体の中での慣性モーメントとは必ずしも一致しないのである。

部品の慣性モーメントは，［物体全体の回転軸からその部品の重心までの距離の2乗に質量を掛けた値］および［その部品自身の慣性モーメント］の合計である。

＊　＊　＊

以上，力学シミュレーションを行うための基本的な道具について解説した。

これらの知識を持っているということと，具体的な事象に対するシミュレータを作れるということの間には大きな隔たりがある。ここから先必要なのは，シミュレートしようとする対象自身についての知識と，大きな問題を小さな問題に分割していく技量なのである。

■現実的なシミュレーションを行うために

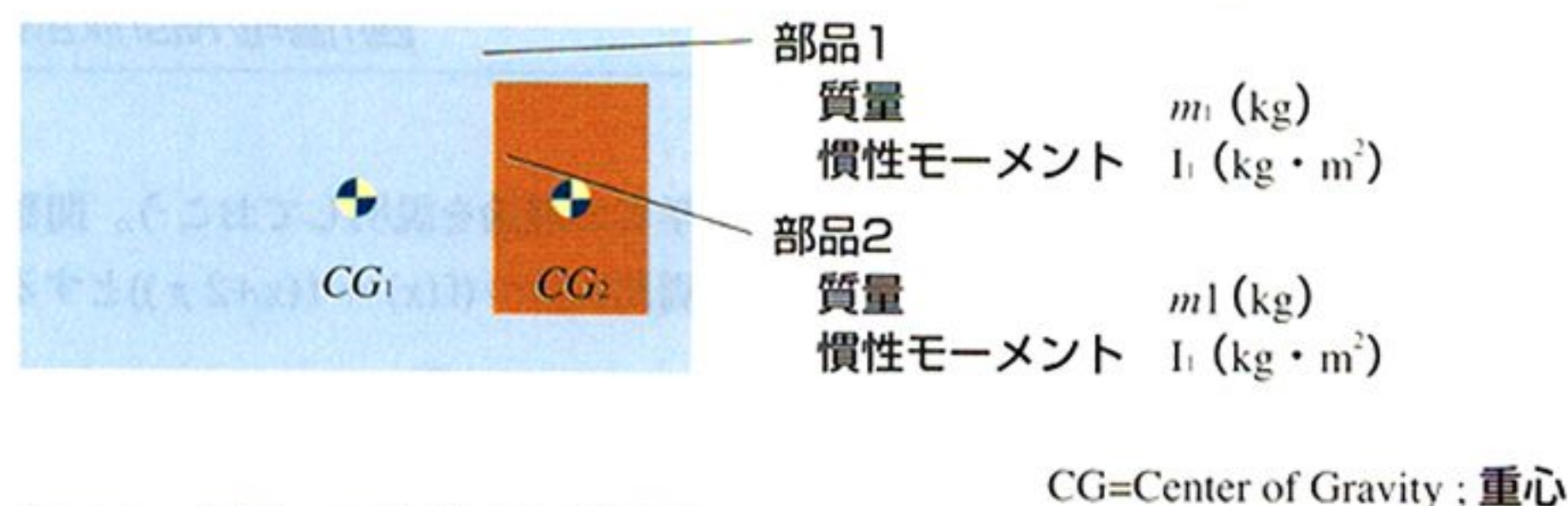
繰り返しになるがゲームはリアルタイムシミュレーションである。だから扱っている事象の複雑さにかかわらず現実的な時間内で処理しなくてはならない。1サイクルの計算処理がΔtで完了しなければ，シミュレーションはそのリアルタイム性

図15　慣性モーメントの合成

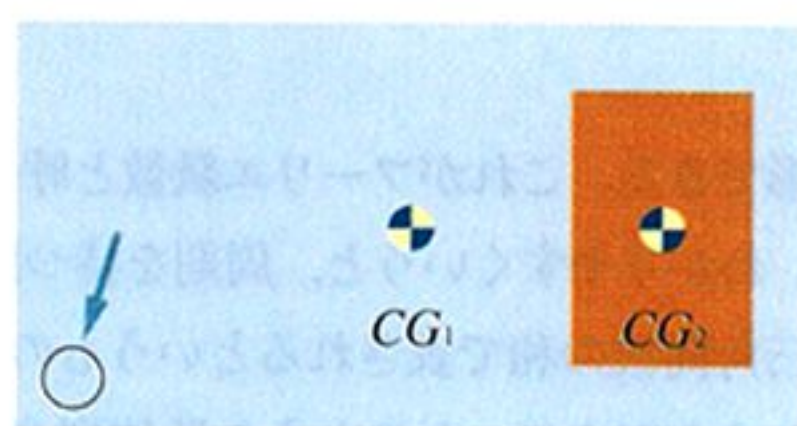
複雑な物体がより単純な部品に分解でき，各部品の慣性モーメントがわかっている場合，物体全体の慣性モーメントは比較的簡単に計算することができる

CG₁　CG₂

部品1
質量　m_1 (kg)
慣性モーメント　I_1 (kg・m^2)
部品2
質量　$m1$ (kg)
慣性モーメント　I_1 (kg・m^2)

CG=Center of Gravity；重心

〔ステップ1：回転中心の決定〕

物体が軸に固定されている場合は，その軸を回転中心とする

CG₁　CG₂

物体が軸に固定されていない場合は，各部分の質量と重心から物体全体の重心を求め，それを回転中心とする

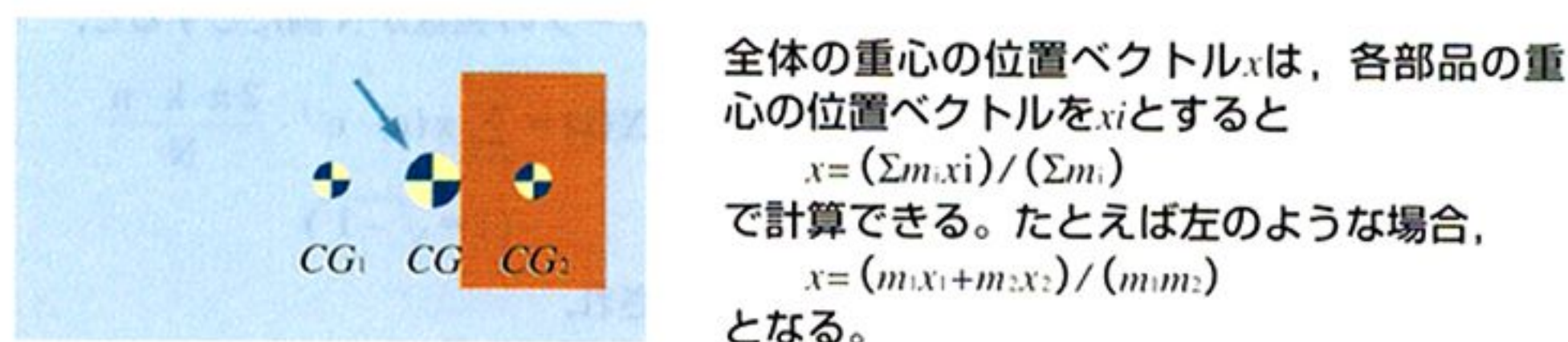
CG₁　CG　CG₂

全体の重心の位置ベクトルxは，各部品の重心の位置ベクトルをxiとすると

$$x=(\Sigma m_i xi)/(\Sigma m_i)$$

で計算できる。たとえば左のような場合，

$$x=(m_1x_1+m_2x_2)/(m_1m_2)$$

となる。

〔ステップ2：慣性モーメントの合成〕

次の合成側を適用する。物体が固定されていてもいなくても法則は同じ

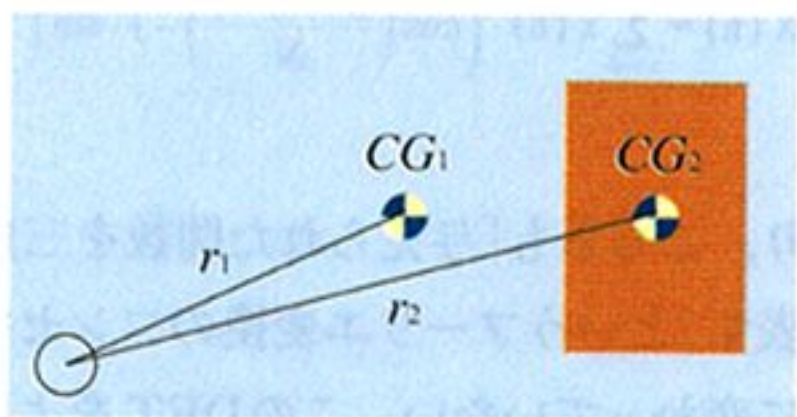
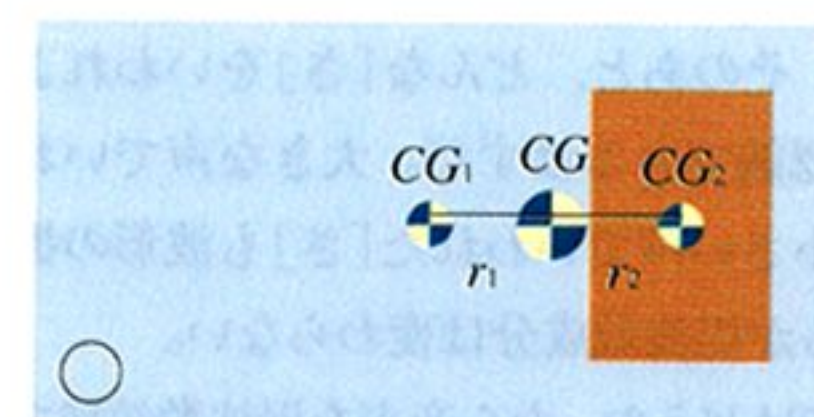
〔各部品の質量m_iに回転中心軸からその部品の重心までの距離r_i2乗をかけた値〕と〔各部分の重心を回転中心とした慣性モーメントI_i〕の合計が全体の物体の慣性モーメントIとなる。

$$I=\Sigma(m_i \cdot r_i^2+I_i)$$

(物体が軸に固定されている場合)

CG₁　CG₂　r_1　r_2

(物体が軸に固定されていない場合)

CG₁　CG　CG₂　r_1　r_2

いずれの場合でも

$$I=m_1r_1^2I_1+m_2r_2^2+I_2$$

で計算できる。両者の違いはr_iの値のみ

を失ってしまい，破綻してしまうのだ。

現状の限られたCPUパワーの範囲内でシミュレーションを成立させるためには，コーディングテクニックも必須だが，そもそもの計算量を減らすほうが効果がある。対象となるものごとを構成する膨大な要素のうち，どの要素を残してどの要素を削るかを見極めなくてはならない。それはモデル化を行う人間の才覚である。

もちろん一発で決める必要はない。最低限のモデルを組み立ててみて，望みどおりの挙動が得られなければなにが足りないか考え，少しずつ要素を足す。無駄な要素が発見されたら勇気を持って削ることも必要だろう。

家庭用ゲーム機クラスでの目標は，非常に雑ないい方ではあるが，「10%の演算量で90%の精度」といったところだ。

出発点から軽さのみを目指してほかのすべてを犠牲にした設計にすると将来性がない。複雑なシミュレーションモデルはCPUパワーが上がってきたときに結実する。システムの基盤には，シンプルで力強くきっちりとした骨格を持たせ，必要に応じて複雑にしていける構造にすることをおすすめする。

■海外ソフトメーカーの地力

ここで唐突だが海外のソフトを話題にしたい。

昔から少なからぬ海外のソフトメーカーが3Dを指向してきた。いまのようにゲーム機が3D指向になりビデオカードが3D対応になるはるか昔，ソフトウェアレンダリングを余儀なくされていた時代からやっているだけあって，3Dに関する蓄積は日本を圧倒しているように思う。そしてそれは昔からシミュレータ指向であることと無関係ではない。

ただ，彼らのシミュレータ指向はやや極端に走る傾向があった。海外の優れた作品は，いつもその時代の計算機能力に比べて無茶な処理をしてきており，妙にフレームレートが低かった。そしてユーザーもそうした低フレームレートのゲームを許容してきたフシがある。

しかしここ最近のハードウェアの劇的な速度向上により，フレームレートの心配をする必要がなくなってきた。つまり，昔から海外でされてきた無茶がもはや無茶でなくなりつつあるのである。

そこで心配になるのが，近い将来，日本の作品が海外の作品に質的に対抗できなくなるのではないかということである。もちろん，現状を全体的に見れば，日本製ゲームのほうがフレームレートの確保やプレイアビリティに気を使っている分，遊びやすい。しかしそれは将来に対してなんの安心も与えてはくれない。

そろそろ，多少の無茶をしつつ将来に備えるという態勢が日本のソフトメーカーにも必要になってきていると思うのだ。

■もっと勉強しなければ

計算機の扱い(OSやプログラミング言語やハードの制御)に長けているだけではよいゲームは作れない。というかそれは必要条件にすぎない。それ以外の世界をどれくらい知っているかが明暗を分けることになる。その傾向は今後ますます加速していくだろう。

面白いゲームを作るための方法論は私には専門外。よくも悪くも技術屋にすぎない自分としては，理論面での強固な土台を提供できる存在でありたい。もっと勉強が必要だ。そして読者諸氏にも，なるべく若いうちにいろいろ学習しておくことをおすすめする次第だ。

音声認識に挑戦してみる

西川善司/Nishikawa Zenji

音声認識……というと意味解析とかの人工知能系の処理を思い浮かべる人も多いかもしれないが，基本になっているのは，音声を適切に分析したあとのパターンマッチングだ。ここでは周波数変換後のデータから解析に挑戦してみよう。

■身近になった音声認識

「音声認識」というキーワードは最近よく耳にする。カーナビやオーディオがボタンを押すことなく話しかけるだけで機能するというあれだ。電話機自体に番号をしゃべったり，登録してある人物の名前をいうだけで相手にかけられる携帯電話も出てきたし，パソコンの分野でもこの技術の応用は盛んで「話しかけるだけで文書入力ができる！」というソフトも出てくるようになった*1。

音声認識がものすごく身近になってきている昨今だが，そのうち「かゆい！」とさけぶと自動的に薬箱から出てきて塗りにくるキンカンとか，「食べたい！」と話しかけると，自ら皮を脱ぎ始めるバナナとかも出てくるようになるのだろうか。

さて，「音声を認識する」というとものすごく難しい次元の話に思えてしまう。実際，認識の手法自体は非常に「難問」とされるテーマであり，世界中の大学や企業の研究機関でいまだ，よりよい方法を見つけるために研究されている状況である。

しかし，音声を取り扱う以上，音声波形に着目せざるをえないことになる。突き詰めていえば音声認識とは「比較される音声波形と，入力音声波形が同一とみなしてよいかどうかの判断」ということになる。これはつまり我々パソコンユーザーにも見なれた「音声データ」同士の比較作業ということでもあるわけだ。

ただし，音声波形というのは同じ人物が同じ言葉をしゃべったとしても同一の音声データになることはまずありえない。たとえばマイクに「おはよう」と2回話しかけPCM録音してできた2つの音声ファイルをバイナリコンペアツールに通してもまったく違うファイルとして判断されてしまうはずだ。その2つの音声波形を波形表示可能な音声ファイルエディタを用いてPCM次元（時間次元）で眺めた場合でも「だいたいは似た感じの波形になっているが細部は異なる……」というのが見た目でもわかる。

それでは，はたして音声認識とはどのような原理で行われているのだろうか。

今回，DTM関連の話題にはそこそこ強い私，西川善司が，無謀にもほとんど未知の領域である音声認識に挑戦してみることにした。はたしてうまくいくのか!?

＊1　「バカ！」と叫ぶと「そんなこといわないでください」となだめてくるカーナビもあるらしい。ちなみに「バカ」と入力して「ソレハアナタデス」と返ってくるのはバンダイのアドベンチャーゲーム「サザンクロス」だ。

■音声認識は周波数領域で

と，ここまででもはやPCMデータ次元，すなわち時間軸次元での音声比較はほとんど無理ということがわかるはずだ。音声波形のパワー（音量）変化に着目し，これをベクトル表現にして，このベクトルを比較対象とする……というのも容易に思いつく方法ではあるが，ほとんどアクセントの位置の違いしかわかるまい。実際それで用が足りる場合も多いのだが，もうちょっとまじめに取り組んでみたい気もする。

そこで考えられるのが，周波数次元での音声波形データの比較だ。ある瞬間，その音はどういう周波数成分を含んでいるのだろうか……この視点で音声を評価していくというわけだ。そこで，図1のような模式が思いつく。

たとえば「さ（SA）」という音は最初にノイジーな音で，そのあと母音となる「あ（A）」がくるわけで，「さ」を細かく分けて，その特徴的な周波数成分の移り変わりをパラメータとして抽出してしまえば，そのあと，どんな「さ」をいわれようと「さ」と認識できるはずだ。大きな声でいわれた「さ」も小さい声でいわれた「さ」も波形の振幅こそ変わるが周波数成分は変わらない。

それではどうやったら音声を周波数領域で分析できるのだろうか。

■フーリエ変換登場

これは過去の本誌Oh!Xに幾度となく登場した「フーリエ変換」という方法を適用する*1。

「フーリエ変換」自体はいまでは実にオーソドックスなデータ解析方法で，実は音声認識以外にも，画像解析や画像認識，はたまた数値演算からデータ圧縮技術までさまざまな分野で適用されている。フーリエ変換そのものについて詳しく知りたい人は一般的な数学書，デジタル信号処理関係の本を参照してもらいたい。

簡単にこのフランスのフーリエ（Fourier）という数学者の理論を説明しておこう。関数f(x)が2πを周期に持つ(f(x) = f(x+2π))とすると，

$$
\begin{aligned}
f(x) &= \frac{a_0}{2} + \sum_{n=1}^{\infty}(a_n \cdot \cos(n \cdot x) + b_n \cdot \sin(n \cdot x)) \\
&= \frac{a_0}{2} + (a_1 \cdot \cos(x) + b_1 \cdot \sin(x)) \\
&\quad + (a_2 \cdot \cos(2x) + b_2 \cdot \sin(2x)) + \cdots \\
&\quad + (a_n \cdot \cos(n \cdot x) + b_n \cdot \sin(n \cdot x)) + \cdots
\end{aligned}
\qquad \cdots (1)
$$

と変形できる。これがフーリエ級数と呼ばれるものだ。わかりやすくいうと，周期を持つ関数はみんな三角関数の和で表されるというものだ。

これをPCMデータのような数値列（離散データ）で表される関数に適用したものが離散フーリエ変換（DFT：Discrete Fourier Transform）だ。データの個数がN個だとすると，

$$
X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-j \cdot \frac{2\pi \cdot k \cdot n}{N}} \quad (j = \sqrt{-1}) \qquad \cdots (2)
$$

で表され，

$$
e^{-jx} = \cos(x) + j \cdot \sin(x) \qquad \cdots (3)
$$

なので，

$$
e^{-j \frac{2\pi \cdot k \cdot n}{N}}
$$

の部分は，

$$
\cos\left(\frac{2\pi \cdot k \cdot n}{N}\right) - j \cdot \sin\left(\frac{2\pi \cdot k \cdot n}{N}\right)
$$

で表されるので結局，式(2)は，

$$
X(k) = \sum_{n=0}^{N-1} x(n) \cdot \left(\cos\left(\frac{2\pi \cdot k \cdot n}{N}\right) - j \cdot \sin\left(\frac{2\pi \cdot k \cdot n}{N}\right)\right) \qquad \cdots (4)
$$

となり，こちらも「与えられた関数を三角関数の和で表す」というフーリエ変換のコンセプトは基本的に変わっていない。このDFTをよりコンピュータプログラム向けに高速化したアルゴリズムが高速フーリエ変換（FFT：Fast Fourier Transform）という手法だ。こちらの高速化の原理についてはOh!X1995年3月号39ページか専門書を参照してほしい。

しかし，ここで疑問に思った人もいるはずだ。フーリエ解析が適用できるのは与えた関数（データ列）が周期関数の場合だけなのに，ランダム波

図1　音声認識の流れ

音声入力 → 周波数分析 → 比較/評価 → 判定
比較/評価 ↓ 周波数データベース

に近い音声データに勝手に起用してしまって正しい結果が得られるのか……という疑問だ。これは確かに痛いところを突いている疑問なのだが，音声解析に限っていえば，人間の音声は数十msというブロックで考えるとほぼ定常波とみなされることが実験的にわかっているそうだ。

ということは，長い音声データも，このくらいの単位でブロック化してフーリエ解析すればその音声データのその区間の周波数特性がわかるということだ。

＊1　Oh!X 1988年8月号p.64，1991年12月号p.76，1994年1月号p.66，1995年3月号p.39&p.44に具体的なプログラムと活用例がともに紹介されている。

■音声分析の基本パラメータ

より具体的な音声認識アルゴリズムを検討する前にここで，どのくらいのサンプリングレートで音声を取り扱うかという議論をする必要がある。

参考文献[1]によれば人間の音声は図2のような分布をしているそうだ。だいたい実用上，普段人間が音声認識に必要としている音声の周波数は200Hz～3kHz程度のようだ。男女の声質の差については200Hz以下の成分の多い少ないの違いだけだそうで，それを無視すれば男女の声も同列に扱うこともできるようだ(実際は難しいようだが)。

ということは6kHz程度のサンプリングレートでもよい……ということになるわけだが，たいていのPCのPCM音源は最低のサンプリング周波数が8kHzなので，これを使うことにする。サンプリング定理からすれば，8kHzのサンプリング周波数では4kHzの音までの再現性は保証されることになる。これならば必要十分といえる。整数演算のしやすさからサンプリングビット数は16ビットを選択，チャンネル数は当然1，すなわちモノラルとする。

これが決まると，前段で少し触れた，FFTにかける際の最小音声ブロック長も決まってくる。サンプリング周波数8kHzということは1秒間に8000個のPCMデータが作られるということである。たとえば50msの小ブロックを考えるとすると8000×0.05=400……つまり400個のデータが50msに相当するということだ。

今回使用するFFTルーチンはごく一般的なクーリーチューキー(J.W.Cooley & J.W.Tukey)法の2の基数のFFTなので，サンプル数は2のべき乗でなければならない。となれば400に近い512(=2^9)個が適当なフーリエ解析単位ということになる。ちなみにサンプリング周波数8kHzのときに512個のデータは512/8000=0.064で，解析単位は時間にして64msということになる。

もちろん，ここでいっているのはあくまで一例なので，解析単位を256個(32ms)としたりするのもいいだろう。どんな結果になるかは実際にやってみるまでわからない。

■FFTとスペクトル

ところで，ある音声データをFFTにかけると，実数部と虚数部の数値列に分かれるのはわかるだろうか(式(4)参照)。

通常，N個の数値列を一般的なFFTプログラムにかけると式(4)でいうn＝0からn＝N-1までの各項のcos成分とj･sin成分の係数がわかることになっている。この係数の大小こそがその波形データの周波数成分の大小に合致する。この，分けられた周波数成分の大小の数値列を一般に「スペクトル」といったりするわけだ。

それでは第何項目の係数が具体的に何Hzの周波数成分量に対応するのだろうか。

フーリエ変換は与えられた波形を三角関数波形に分解するもの……ということもできるわけで，それぞれの項がある周波数の三角関数波形を表していることになる。この「ある周波数」は「基調波」と呼ばれる波形の周波数の整数倍で表され，これは，1秒間当たりの波形データ数をFFTのポイント数で割ると求められる。今回のケースを例にとると話はいっそうわかりやすい。

サンプリング周波数8kHzなので1秒間当たりのデータ数は8000個だ。これを512ポイントのFFTにかけるので結局，基調波の周波数は8000/512=15.625[Hz]ということになる。これで，第1項目の周波数は15.625Hzということがわかり，第2項目は31.25Hz，第512項目は8kHzを表すことになる。このようにして，たとえばこの例の場合，第2項目の係数の絶対値が大きければ31.25Hzの成分が強いということになるのだ。

いい忘れるところだったが，cos波はsin波に対して90度位相が遅れた成分を表しているので，純粋なスペクトルを求める今回のような場合は，三角関数の加法定理，

$$a \cdot \sin\theta + b \cdot \cos\theta = \sqrt{a^2 + b^2} \cdot \sin(\theta + \alpha)$$
$$\sin\alpha = \frac{b}{\sqrt{a^2 + b^2}}$$
$$\cos\alpha = \frac{a}{\sqrt{a^2 + b^2}}$$
$$\alpha = \arctan\left(\frac{b}{a}\right)$$

を使ってAcos (x) + jBsin (x) の形態を取っているFFTの結果をsin成分だけの式へ合成変換処理する必要がある。

■音声認識の具体例を考える

だいたい，手法が見えてきたところで，どんなものを今回作るか……という議論に入ることにする。いきなりしょっぱなから音声入力ワープロとかを作るのはいくらなんでも無理だ。そこで簡単な例として2つの音声ファイルの類似性を評価するプログラムを作ることに決めた。

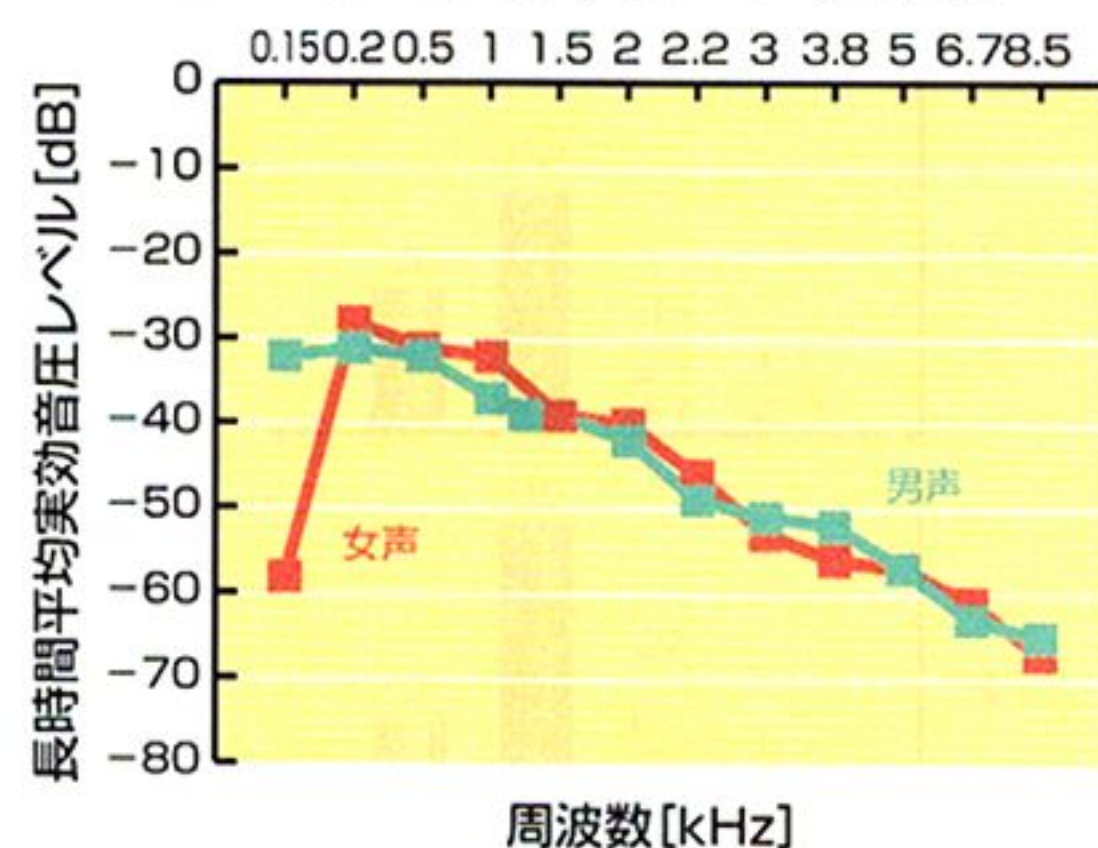



図2　音声波形の長時間平均スペクトル分布

仕様としてはこんな感じだ。

たとえばある人物が「こんにちは」を2回しゃべってそれぞれを録音したとする。この録音作業でできた2つの「こんにちは」の音声ファイルをこのプログラムで比較すると「似ている」と判断する。もちろん，この人物が「おはよう」としゃべって録音したファイルと，先ほどの「こんにちは」と比較した場合は「似ていない」と判断できなければならない。

さて，その実現方法だが，簡単に流れを考えるとこんな感じになるはずだ。

1）比較する2つの音声ファイルのスペクトルを求める。

↓

2）スペクトルのばらつきを比較して一致，もしくは近ければ「似ている」と判断

理論上はこれだけなのだが，これだけでは，まずうまくはいかない。

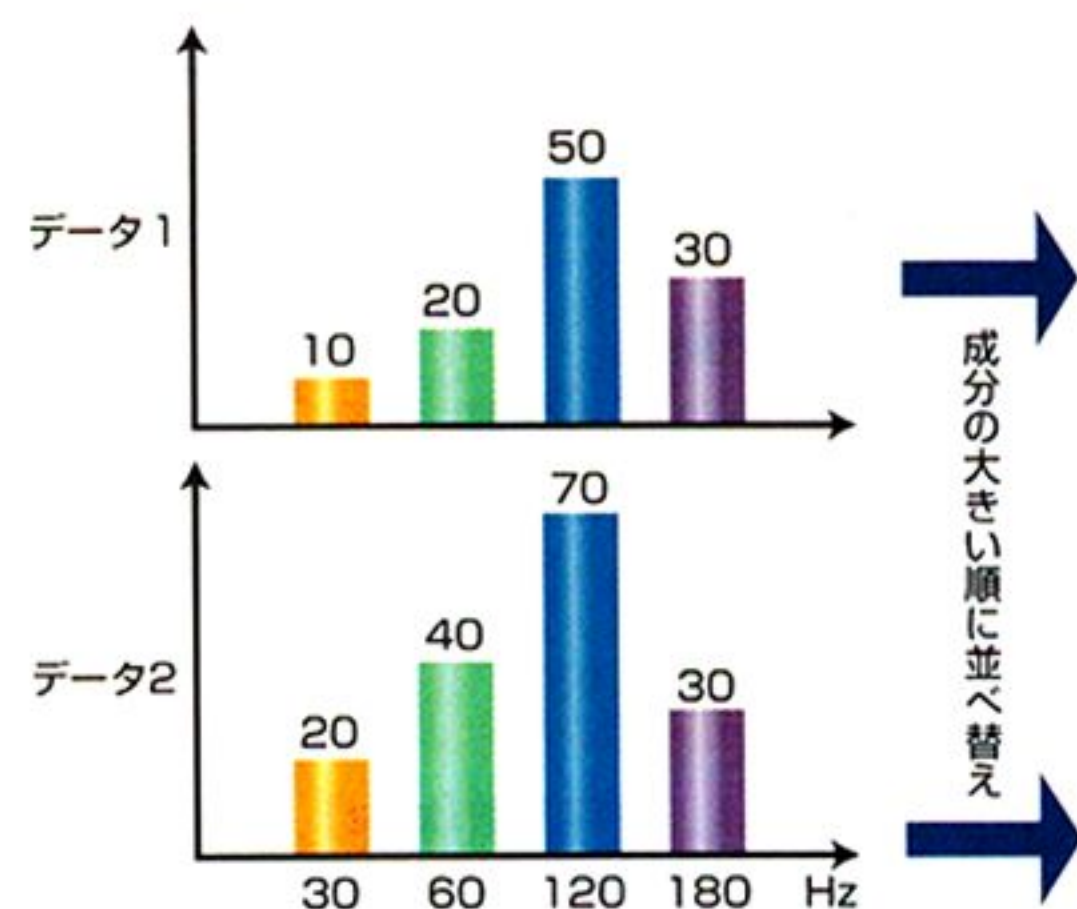
仮に同じ人が音声「こんにちは」を2回録音したとしても十中八九，その2つの音声データの長さが一致することはない。つまり，長さの違う音声データ同士の比較の際の辻褄をあわせなければならないのだ。

この「辻褄あわせ」は音声認識を議論する場合には避けては通れない道のようで参考文献にもさまざまな方法が記載されていた。

もっとも単純なのが，時間軸上での波形伸縮をやってしまい両者のデータ長を揃えてしまう方法だ。しかしこの方法では音声の特徴が伸縮処理によって欠如する可能性がある。

そこで今回は手間がかからないうえに，それなりの効果がありそうな別の方法を考えてみることにした。

それは，連続音声を音節単位でブロック分けし，この単位での比較を行うというものだ。必ずしも理想どおりにはならないが，たとえば「こんにち

図3　今回のプログラムの認識方式例

は」という音声を「こ」「ん」「に」「ち」「は」と5つの音節に分け，比較対象も同様に音節に分解し，音節同士をスペクトル比較するというものだ。

問題となるのは，この音節への分け方の手法なわけだが，これは音声波形のパワーに着目する原始的な方法をとることにした。つまり，突然波形が立ちあがったり，アクセントが強まったりしたところを音節の始まり，とする手法だ。それだけではなく，一応，波形のギザギザを図学的にも捉え，ギザギザの出方の規則性が変化したところにも注目するようにした(プログラムリスト〝anaeva.cpp〟の関数splitwave()のところ)。

■スペクトルの比較方法

スペクトルの比較方法は，比較に使用する2つのスペクトル同士の差を周波数帯1個ずつ順番に計算していく方法でもよかったのだが，比較時の演算量を少なくするために少しだけ工夫を凝らしてみた(プログラムリスト〝anaeva.cpp〟の関数analyze()のところ)。

スペクトルをFFTによって普通に求めたあと，このスペクトルを構成する周波数をその強さ順に並び替えてしまうのだ。たとえば30Hz，60Hz，120Hz，180Hzの強さがそれぞれ10，20，50，30だとすれば120Hz，180Hz，60Hz，30Hzと並べ替えてしまうということだ。

その音声波形の特徴となる周波数ほど強く，並べ替えの時点で前のほうにくる。逆に，この並べ替えで後ろのほうにきた周波数は認識の際には取り扱い不要の無視できる周波数帯ということができる。つまり，無視できるスペクトルは無視してしまえというわけだ。

さて，512ポイントFFTの場合，基調波からサンプリング周波数と同一の周波数帯まで512個のスペクトルが抽出できるわけだが，そのうちの全部を比較対象(並べ替え対象)とするのはナンセンスだ。冒頭でも少し触れたように，認識するのに必要十分な帯域はたかだか3kHz程度までである[*1]。FFTは512倍波までのスペクトルを求めるが，比較演算時には300Hz～3kHzの範囲を超える周波数帯については無視することにしている。

実際の2つのスペクトルデータの比較の方法だが，事前に両者ともにスペクトルデータを強い順に並べ替えているので，その並び方から偏差を計算する……という方針をとってみた。

たとえば，スペクトルが30Hz，60Hz，120Hz，180Hzで構成され，音声データ1のそれぞれの強さが10，20，50，30，音声データ2のほうは強さが20，40，70，30だとする。両者，周波数をその強さ順に並べ替えると，

データ1　　120Hz 180Hz 60Hz 30Hz
データ2　　120Hz 60Hz 180Hz 30Hz

ということになる。

この並び方の違い(偏差)を評価して採点するというわけだ(図3参照)。その偏差がゼロに近ければ近いほど，両者は「似ている」ということになる。ほかにもさまざまな比較方法があるのだろうが，今回はこういう手法を使っている。

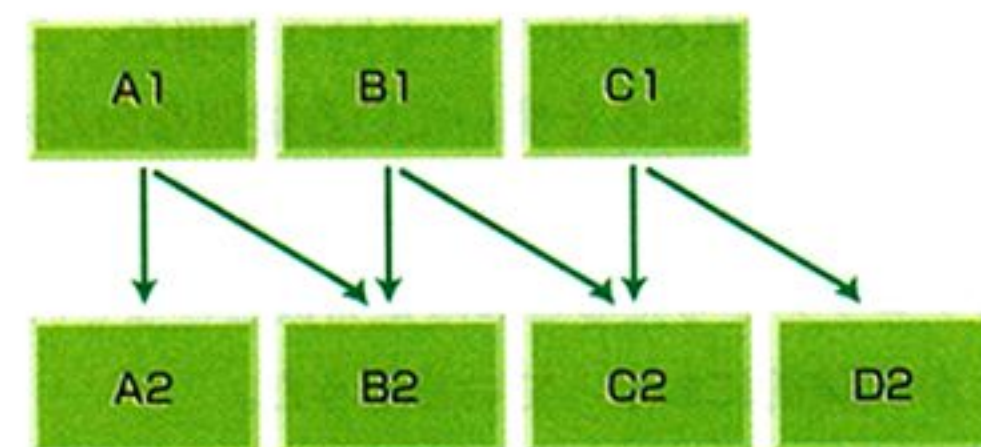
なお，今回作ったプログラムではFFTを施してできたスペクトルを300Hzから3kHzまで20Hzごと，135個の周波数帯に分けてスペクトルの強度を算出し，これを強さ順に並べ替えて強度トップ64を求めている。2つの音声データのスペクトル要素のうち，このトップ64同士を比較して算出された偏差が比較結果ということだ(プログラムリスト〝anaeva.cpp〟の関数compare_spectrum()のところ)。

＊1　場合によってはそれ以下でもいいこともあるようだ。

■個数の違う要素間の比較について

今回試作したプログラムの流れを整理してみる。

まず，スペクトル比較前に音声波形を適当な音節片に分割している。この分割された音節を64msごとにFFTでスペクトル解析し，この音節単位でのスペクトル比較を行う。すべての音節同士のスペクトル比較を行い，総合偏差がどのくらいかで類似度を算出する。

図4　音節片の数が違う場合の比較方法

ところで，最初のこの音節分割が正しいという保証はない。同じ言葉「こんにちは」でも冒頭の歯切れが悪ければ(?)，片方の音声データは「こん」「ち」「は」と分解されるかもしれない。

こういったケースを想定して，分割を行って，両者の音節数が違った場合でもある程度適切な比較処理が行える工夫をしてある。

たとえば，2つの音声データの音節が「A1 B1 C1」と「A2 B2 C2 D2」という3つと4つに分かれてしまった場合，先頭から互いに1対1で比較するとD2の比較相手がいないことになる。もしかしたらA2がさらに2つに分かれるべきで，D2はC1と対応する音節かもしれない。

こういう状況に対応するために，図4のような音節の少ないほうが，多いほうへ複数回比較するような方法をとってみた。大幅に音節数が違う場合は問題が出るが，少々の差ならばこれで音節片分割の誤差を修正できる。

実はこの比較方法は，スペクトルの偏差を求めるときにも使っている。比較する2つの音節の音長同士が等しいことは稀であり，そうなれば音節を64msで分割してから求めたスペクトルブロックの数も異なることになる。

このような1対1の比較が不可能なときは，複数回の比較処理が入るわけだが，結果的にはその複数回の比較処理のうち，もっともスコアのよい値を最終的な結果として採択している。

ここまでをまとめると図5のような感じになる。

■プログラムの解説

作成したプログラムはWindows95/98/NT(Win32)のDOSプロンプト用のものでディスク1のCD-ROMのディレクトリ「ZWFC」に収録されている。動作奨励環境としてはCPUがPentium100MHz程度，メモリが32MBだろうか。もっともリアルタイム性を要求するプログラムではないので486以下でも動作はできると思うが。

ところでCD-ROMに収録されているプログラムは編集のスケジュールの関係でやや不完全なままで収録されてしまったので，できれば最新版を，

ftp://ftp.z-z-z.gr.jp/temp/zwfc.lzh

からダウンロードしてほしい。

どうしてもインターネット環境がないという読者の方はソースリストのほうを少々修正を施してもらいたい。変更するところはほんの少しだけ。

ソースリスト“anaeva.cpp”の冒頭の#defineの部分をリスト1(list1.txt)のように修正してコンパイルしなおしてほしい。いうまでもないがコンパイルにはMicrosoft Visual C++が必要だ('98年9月現在の最新版はVer.6.0)*1。

具体的には「ファイル」-「ワークスペースを開く」メニューから“veval.dsw”を開き，ソースリストを修正したあと「ビルド」メニューから「リビルド」を選択すればコンパイルが完了する。

プログラム名は「ZWFC.EXE」。

使い方は簡単で，「サウンドレコーダー」などで，

サンプリングビット数16bit
サンプリング周波数8kHz
モノラルチャンネル

で録音したWAVファイルの2つのファイル名を，

```
A>ZWFC -Z filename1 filename2
```

としてパラメータで与えるだけ。比較結果は100点満点で表される。100点に近い高得点ならばその2つの音声ファイルは「似ている」といえ，得点が小さければ「似ていない」ということになる。

採点はスペクトル次元での採点結果のほか，音節分割をした際の，分割個数比にも着眼して採点している。

1 ソースリストのファイル名は“.cpp”だが，使っているのはC言語である。

■できばえはどうだったか

今回作ってみたZWFC.EXE……何通りものテストケースで試してみた手応えだが，完成度としては今一歩というところ。似た音声同士だとちゃんと100点に近い高得点を出すし，明らかに違う音声同士では得点は低くなる。ところが，ちょっと意地悪なデータ同士を比較すると，途端に異常な結果を返してくる。

まず，問題として挙げられるのは同一人物の音声だと，たとえいっていることが違っていても高得点になりやすい傾向があるという点。逆に違う人物の言葉だと，たとえしゃべっている言葉自体が同一でもスコアが低くなってしまう。実際の市販の音声認識ソフトでは，こうした状況に対応できるように，同一の言葉でも，複数の認識パターンを持っているものなので，「2つの音声の類似性を判断する」というだけの今回のプログラムではまぁ致し方ないところではあるのだが……。

もうひとつは，比較する音節の分割数が異なると，極端な点数が出ることがあるという点。たとえば入力音声1「こんにちは」と，ここに含まれる「に」の部分を切り出して作った入力音声2を比較したとするとなんと100点近い点が出てしまうのである(ガーン)。これは実は本文中で触れた「音節数が異なる場合の比較のための工夫」が裏目に出てしまった結果のようだ。

この例でいけば，入力音声2「に」は，入力音声1の文節「こ」「ん」「に」「ち」「は」のそれぞれと比較されることになる。音節数は1：5の関係なのだが，「に」だけの音声データ2は「こんにちは」の音声データ1と同列のものとして比較されることになるのだ。ここに問題がある。

データ2「に」とデータ1の文節「に」を比較したときに完全一致することになり，ほかの比較結果よりもダントツで偏差が小さいので，これが最終的な比較評価値となってしまうのだ。ほとんどバグみたいなものだが，これは，ソースリスト“anaeva.cpp”の中の180行付近，

```
if (sc<100)
{
    sb= ((float) blkn1/(float) blkn2)*50;
    sc+=sb;
    if        (sc>100)   sc=100;
}
```

の後ろに，

```
else
{
    sc *= (float) blkn1/(float) blkn2;
}
```

を追加するとそれらしい結果になる。要するにスコアに音節数の比をかけてしまうのだ。これならば本当に音節数まで完全に一致したときはちゃんと高得点のままになるし，音節数が異なればその分，点数は低くなる*1。

このほか，問題点とまではいわないまでも改善点はいろいろとありそうだ。

たとえば今の状態では類似性評価に音節の個数にしか着目していないが，各音節の長さの比などにも着目したりする必要があるだろう。

また，スペクトル個数の違う音節の比較の際，もっとも偏差の少ない演算結果をスコア要素として採択するが，その後の比較に，それ以前の採択結果をまったくフィードバックしていないことも改善の余地がありそうだ。

たとえば，「ぷぅ」と「ぅあ」という音節を比較した場合があったとして，両者ともに「ぅ」という音韻を持つが，その出現順序が両者では異なる。しかし，このプログラムの現在のバージョンではこの登場順序は評価せずに，単純に一致する音韻を発見したとして偏差を計算してしまう。

この部分は「2つの音声の類似性評価」という今回のプログラムの趣旨にはそれほど影響しないのかもしれないが，本格的な「認識」を行うためには無視できない問題点となるだろう。

*1 blkn1≦blkn2なので100点より大きくなることはない。詳しくはプログラムリスト参照のこと。

リスト1

```
//定数
#define VEVAL           1              //1だとエラーメッセージ表示
#define NOISE           1024           //ノイズとみなすデータレベル
#define PDRATIO         1.5            //音節分割パラメータ
#define HPEAKRATIO      1.5            //音節分割パラメータ
#define LPEAKRATIO      0.25           //音節分割パラメータ
#define BLKBUFSIZE      100            //音節分割パラメータ
#define LOG10_2         0.3010299957   //log10(2)
#define MTRGN           512            //最小解析サイズ[bytes]
#define SMPFRQ          8000           //サンプリング周波数[Hz]
#define ARES            20             //解析周波数解像度[Hz]
#define MINFRQ          300            //スペクトルの最小有効周波数[Hz]
#define MAXFRQ          3000           //スペクトルの最大有効周波数[Hz]
#define NOFBEST         64             //比較対象にするスペクトルの数
#define SPCT_HDRSIZE    16             //スペクトルデータのヘッダサイズ
#define PENALTY         0              //認識パラメータ
#define MUL             1              //認識パラメータ
#define SCMUL           1              //認識パラメータ(スコア算出用重み定数)
#define ACKRES          64             //認識パラメータ(ACKRES<=NOFBEST)
```

図5 今回のプログラムの大まかな処理の流れ

・参考文献[1]
「音声・画像工学」/中田和男・南敏共著/(株)昭晃堂
・参考文献[2]
「確率モデルによる音声認識」/中川聖一著/(社団法人)電子情報通信学会
・参考文献[3]
「音声・聴覚と神経回路網モデル」/甘利俊一監修・中川聖一・鹿野清宏・東倉洋一共著/(株)オーム社
・参考文献[4]
「ディジタル信号処理の基礎」/樋口龍雄著/(株)昭晃堂
・参考文献[5]
「基礎解析」/高橋陸男・大嶋勝・永倉安次郎共著/数研出版株式会社
・参考文献[6]
Oh!X 1988年8月号p.64，1991年12月号p.76，1994年1月号p.66，1995年3月号p.39&p.44

Hardware Geeks

杖を鼠にする方法

Text：桑野雅彦／Masahiko Kuwano

マウス，おそらく大半の人が毎日使っている道具だろう。マウスのみならず，トラックボールやグライドポイントなど，ポインティングデバイスにはさまざまな種類のものがある。ではその仕組みはどうなっているのだろうか？　マウスのメカニズムを見てみよう。

マウス操作はやさしいか

先日，雑誌を見ていたら手の操作が思うにまかせないので，足でマウスを操作しているという写真が目にとまりました。少々ショックを受けて試しにやってみるとこれが非常に難しいのです。

マウスをつかむ，マウスを思った向きに動かす，マウスのボタンを押す，マウスのボタンを押したまま移動するという基本動作に加え，思った位置で静止させること，ボタンを押す間動かないようにするといった，日頃大して気にもとめていなかったようなことが，面倒きわまりないことになるのです。

改めて考えてみると，手で操作しているときでも，隣のメニューをクリックしてしまったり，ダブルクリックしたのに認識してくれなかったり，Windows 95のスタートメニューからツリーを降りていこうとしてうまくいかずにイライラさせられるといったことも頻繁に起こります。ドラッグしていたら途中で指が離れて，やり直したり，違うアプリケーションが受け取ってしまったなどということもありました。

改善策は？

「つかむ」ということからどうしても離れられないマウスを足で操作するということは不可能ではないとはいうものの，簡単ではありません。トラックボールにしても，何度も一定方向に回すのは案外面倒なものです。

もっと簡単に使えそうなものはないかと，パソコンショップを歩き回っていたら，ジョイスティックが目にとまりました。

ジョイスティックは傾ければその方向に動き，手を放せば元に戻り，止まるというインタフェイスが普通です。昔はジョイスティックといえば単なる四角い箱から棒が1本突き出したような，味もそっけもないものしかありませんでしたが，ゲームの多様化と高度化のおかげで本格的なスティック型にとどまらず，車のハンドルのようなものや，十字ボタンのいわゆるゲームパッドタイプなどもあります。どれも同じインタフェイスというところもハード屋としてはうれしいところです。

考えてみれば，ノートパソコンでは中央部分にスティックタイプのポインティングデバイスを設けているものも確かにあります。傾ければ動く，離せば止まるという点を見てもジョイスティックと変わるところはありません。本家ともいえるジョイスティックでポインティングするというのもそれほど変なことではないでしょう。

今回は，もうひとつのポインティングデバイスともいえる，ジョイスティックをマウス化するアダプタの実験をしてみることにしましょう。

ジョイスティックインタフェイスの調査

ジョイスティックとマウスの間を取り持つため，両者のインタフェイスを調べておかなくてはなりません。まず，ジョイスティックインタフェイスを見ていきましょう。

ジョイスティックインタフェイスとして有名だったのは4方向（上下左右）スイッチとAボタン，Bボタンをつけた，いわゆるアタリ仕様というものです。スティックがある方向に傾いたか否かをスイッチで知るだけの仕様ですから，45度ずつでしか傾きを通知できませんが，シューティングゲームなどではこれでも十分楽しめました。

これに対して，互換機のジョイスティックポートはスティックの傾きをアナログ的に知ることができるようになっています。この仕組みについてはLinuxのI/Oポートプログラミングミニハウツーに記載がありました。スティックのX方向とY方向に可変抵抗器（ボリューム）がついたようになっており，可変抵抗器の片側は+5Vに接続されているということです。ジョイスティックインタフェイスでは，CPUがジョイスティックポートを叩くと抵抗値に応じた幅のパルスが出るような回

今回製作したジョイスティック－マウス変換器

今回データを取ったジョイスティックとジョイパッド

表1　ジョイスティックのピン配列

ピン番号	信号名	購入したスティックの配線
1	5V	5V
2	SW1	Aボタン
3	X1	X方向（左：0，右：100KΩ）
4	GND	GND
5	GND	N.C.
6	Y1	Y方向（上：0，下：100KΩ）
7	SW2	N.C.
8	5V	N.C.
9	5V	N.C.
10	SW4	N.C.
11	X2	N.C.
12	GND	N.C.
13	Y2	ベース（上：0，下：100KΩ）
14	SW3	N.C.
15	5V	N.C.

路を組んでおいて，このパルス幅をCPUで計測するという仕組みです。

では，この位置検出用の可変抵抗器の値やコネクタのピン配置などはどうなっているのでしょう。WWWで探してみたのですが，あったのはコネクタのピン配置だけで，可変抵抗の抵抗値や，どちらに傾けると抵抗値が増加するのかといったことが書かれているものにはめぐりあえませんでした。たぶん，業界では常識ということなのでしょうが，値がわからないことには変換回路の設計のしようがないので，とりあえずジョイスティックとジョイパッドを買って調べてみました。

先にWWW経由で調べておいたピン配置と，今回購入したジョイスティックを調べてみた結果をまとめたのが表1です。ジョイスティックポートは最大4個のアナログ入力（X1,Y1,X2,Y2）と4つの接点入力（SW1～SW4）が接続できるようになっています。購入したジョイスティックではX1,Y1がそれぞれX方向，Y方向の傾きに，SW1がAボタン（人差し指で操作するトリガボタン），SW2がBボタン（親指で操作するボタン）に割りつけられていました。購入したスティックは特殊なものではないので，どのスティックも同様になっていると推定されます。

アナログ入力用の可変抵抗器の値をテスターであたってみたところ，100KΩのものが使われていました。抵抗値はX方向は左いっぱいに倒したときが0Ω，右に倒したときが100KΩとなっており，中心にあるときは50KΩです。Y方向は，上側が0Ω，下側が100KΩになるようになっており，やはり中心にあるときは50KΩでした。トリガボタンはスイッチをONにしているとGNDと接続，OFFにしていればオープンという仕様です。連射機能も含めて整理すると内部は図1のようになっていると考えられます。

続いてジョイパッドについても調べてみました。ジョイパッドのほうは，単純な抵抗だけではなさそうなので，分解してみたところ，十字ボタン部分の回路は図2のような回路になっていました。図はX方向分だけですが，Y方向についても同じ回路になっています。

パッドを逆向きに持っても使えるようにするための反転スイッチがあるため，ちょっと面倒な感じですが，要するに通常はQ1がONになっているため，R2が短絡されたようになり，出力はR1（47KΩ）の抵抗が+5Vとの間につながっているように見えるわけです。右SWが押されるとQがOFFになり，R1とR2の合成抵抗である94KΩがついたようになります。また，左SWを押すと+5Vと直結されるため，0Ω相当となります。

ジョイスティックと完全互換にするならR1やR2は50KΩにすべきなのかもしれませんが，入手性がよいこと，多少の誤差はソフトで修正可能であるということで47KΩが使われているのでしょう。

PS/2インタフェイスの実現

マウスインタフェイスとしてはシリアルポートを使う，シリアルマウスもあることはありますが，やはりPS/2マウスが主流でしょう。

しかしこのPS/2マウスのインタフェイス仕様というのがなんとも不思議なものなのです。PS/2ではデータ転送用にDATAとCLKの2本の信号があり，これを本体と周辺機器（マウスやキーボード）が双方向で使用します。通常は両方とも'H'レベルになっており，本体側がデータを出力したいときにはCLK信号を，マウスが側がデータを送りたいときにはDATAを'L'にして相手の反応を見て，自分が信号ラインを使えるようであればCLKとDATAラインを使って相手にデータを送るようになっています。

COMポートに接続するシリアルマウスや，X68000のマウスのように単純な非同期シリアル通信で行ってもなんら問題はないはずで，なぜIBMがこのような奇妙な仕様にしたのかは不明です。このインタフェイスをハード的に実現するのは不可能ではありませんが，非常に面倒です。このため，PS/2キーボードマウスにはワンチップマイコンが入っており，マウスの移動方向やボタンの状態の読み込みや本体側とのデータ転送作業を処理しています。

今回作成しようとしているアダプタの場合，ジョイスティックの傾きやボタンの状態を検出して，本体にはPS/2インタフェイスを通して，あたかもマウスであるかのようにふるまう必要があります。このような処理を行うのにはCPUを使うのがもっとも楽なのですが，汎用CPU＋ROM+RAMという構成では配線量が多くなりすぎます。最近ではPICシリーズのような，エンドユーザーでもプログラム可能なワンチップマイコンもありますが，入手可能なパーツショップはまだ限られています。

そこで，もう少し簡単な実現方法として市販のマウスに入っているワンチップマイコンを取り出して利用することにしました。つまり，マウスから取り出したワンチップマイコンをPS/2インタフェイスチップとして使おうというわけです。

この方法をとった場合，ホストとインタフェイスするソフトウェアはすでに完成しているわけですから，話は比較的単純です。あとはいかにワンチップマイコンをだますかということになります。

マウスの仕組み

では，マウスのワンチップマイコンはどのようにして移動方向やスイッチ情報を得ているのでしょうか。一般的なボールが回転する2ボタンマウスの基本的な構造は図3のようになっています。2つのフォトインタラプタによってX方向，Y方向それぞれの移動方向と移動量を，2つのスイッチがそれぞれ右クリックと左クリックの検出用として用意されています。

フォトインタラプタ部分には図にあるような，円板にスリット状の穴を開けたものが取り付けら

図1　購入したジョイスティックの回路

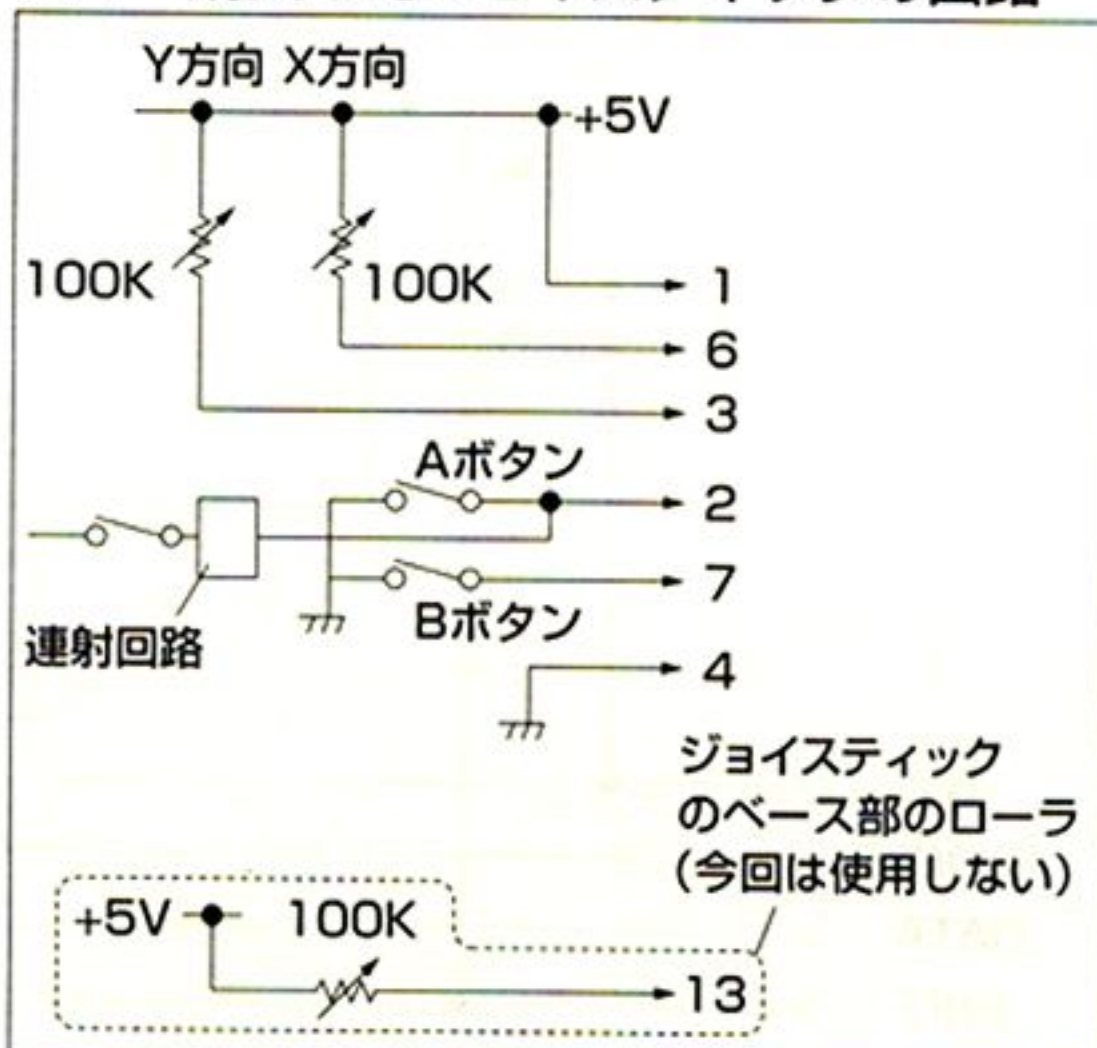
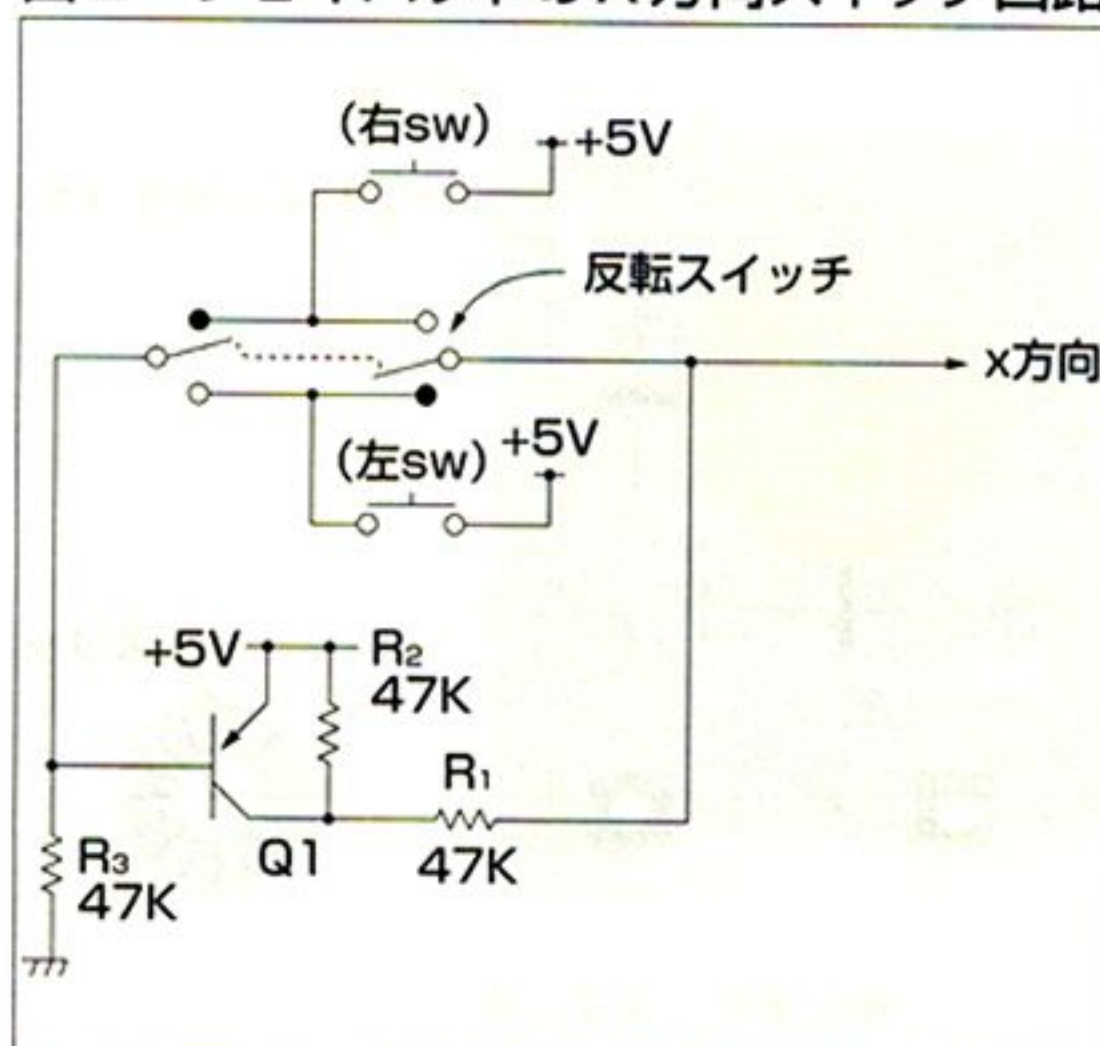



図2　ジョイパッドのX方向スイッチ回路

れており，回転方向や移動距離を検出するようになっています。

マウスの動き検出はこのフォトインタラプタ部分の動作が鍵になります。この部分の仕組みを図4〜図6に簡単に書いてみました。

フォトインタラプタ部分には図4のように2つのセンサが並んでいます。これに対してセンサの幅の2倍の大きさの穴が開いている板を移動させてみましょう。フォトインタラプタ部分の回路は図5のようになっており，穴が開いているとLEDの光がフォトトランジスタに当たるため，フォトトランジスタがONになり，穴が開いていないとOFFになります。マウスの移動量は，この穴によって発生するパルスの数を数えればよいわけです。

では，変化方向はどうすればわかるのでしょう。これはセンサが2つあるところがポイントです。2つのフォトトランジスタのうち，左側をA，右側をBとして，板が右に移動するときを考えてみます。すると，フォトトランジスタのON/OFFの変化はまずA側で発生して，次にB側が変化するという順になります。左に動くときはこの逆で，まずB側が変化して次にA側が変化することになります。

これを波形にしたのが図6です。左半分が右方向移動のとき，右半分が左方向への移動のときの動作波形を示しています。

ここで，Aの立ち上がり（ONからOFFへの変化点）でBの信号を見ると，右方向に移動するときはBはON状態，左に移動するときにはOFF状態になっています。これによって移動方向が判断できるというわけです。このフォトインタラプタのところの信号を切断して同様のパルス波形を与えてやればマウスが動いたようになるはずです。

ちなみに，穴の幅および穴と穴の間隔をセンサのピッチの2/3や2/5にしても同じような波形を得ることができますので，試してみてください。

図3　マウスの回転検出部

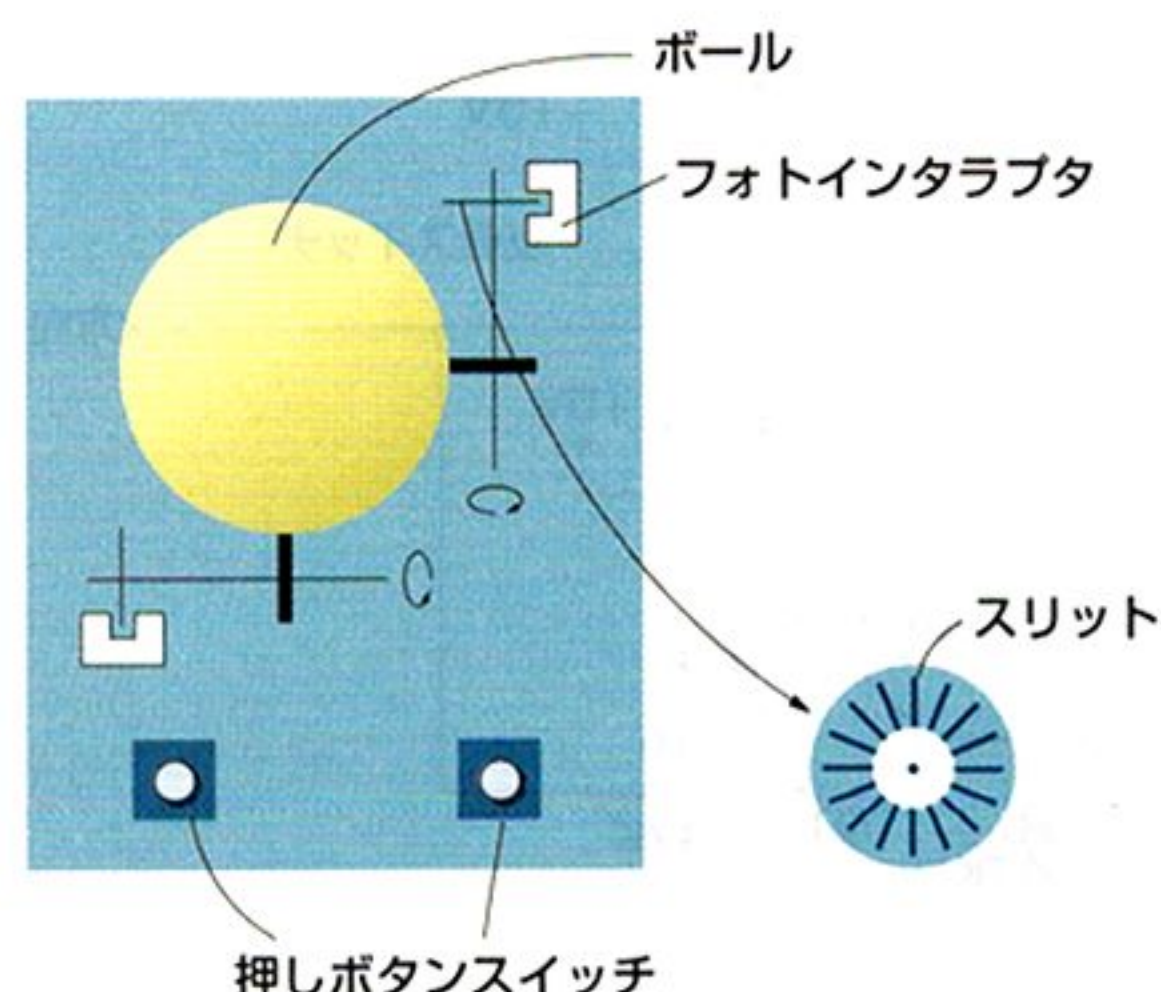



図4　検出部の構造

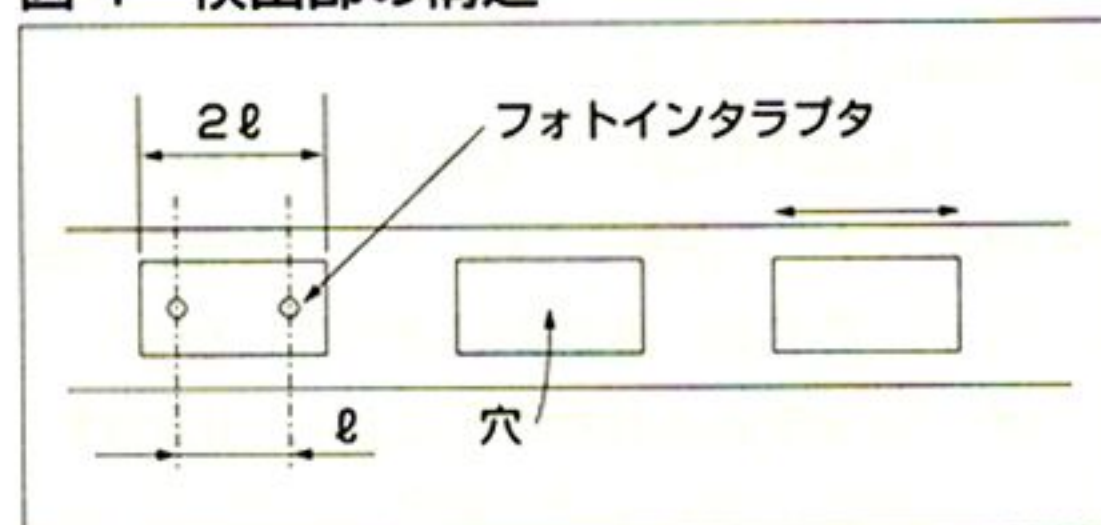



図5　フォトインフラプタ部の構造

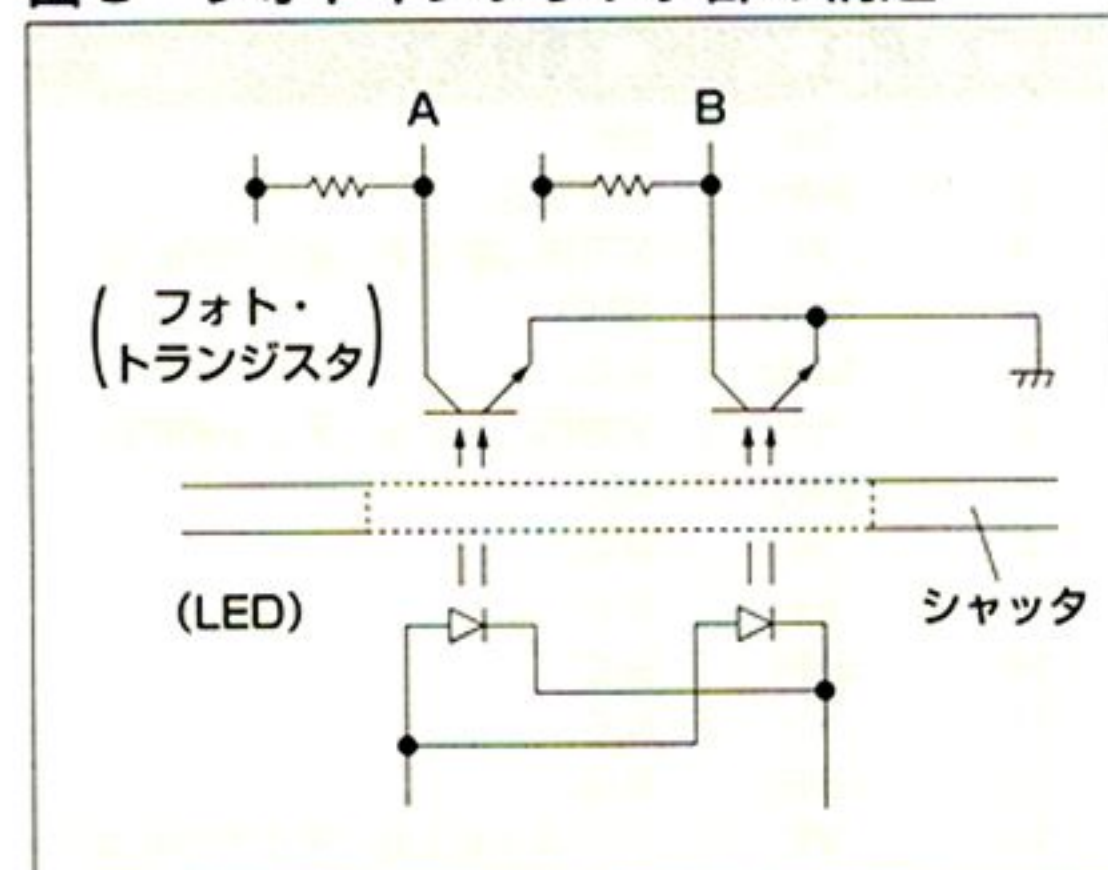
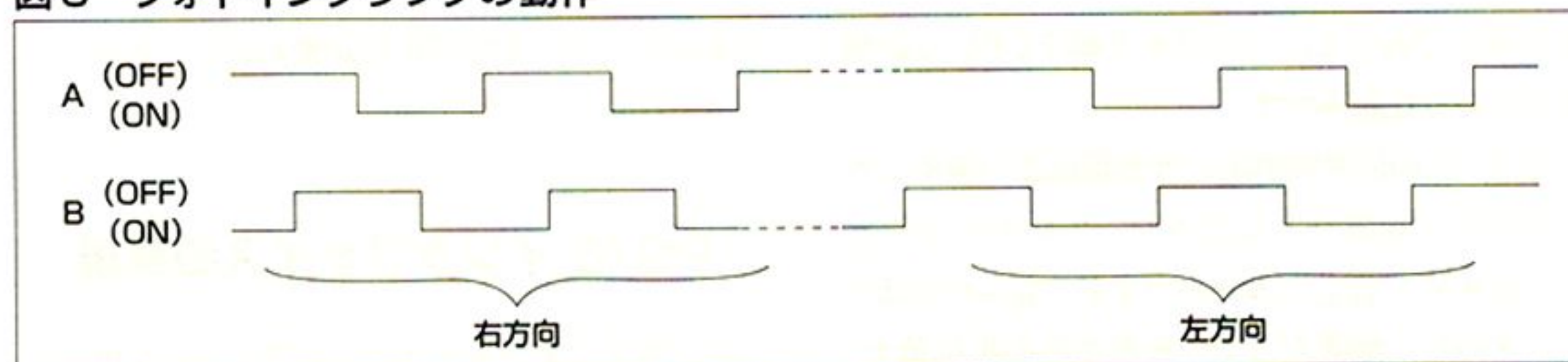



図6　フォトインタラプタの動作

マイクロソフトマウスをばらす

どこのメーカの製品でもマウスの基本的な構造は同じようなものですが，中のワンチップマイコンなどは微妙に異なります。今回は比較的どこでも手に入れやすいであろうということで，マイクロソフトの2ボタンマウスから部品を取り出すことにしました。

まず，マウスを分解します。底面の滑りをよくするためのシールをはがすとネジがありますので，これをはずすと，上側がふたのように取り外せます。フォトインタラプタ部分のメカ部分が固定されていますが，ひねって取ると基板がはずれます。

この基板の回路を調べたところ，図7のようになっていました。図の左端部分がPS/2コネクタ部分で，中央のHM8450APと書かれたICがワンチップマイコンです。回路自体は非常に単純で，X方向，Y方向それぞれの向きのフォトインタラプタと押しボタンスイッチが直結されているだけです。

少し注意が必要なのは，押しボタンスイッチがONになると+5Vとつながるという点です。ジョ

図7　マイクロソフトマウスの内部回路

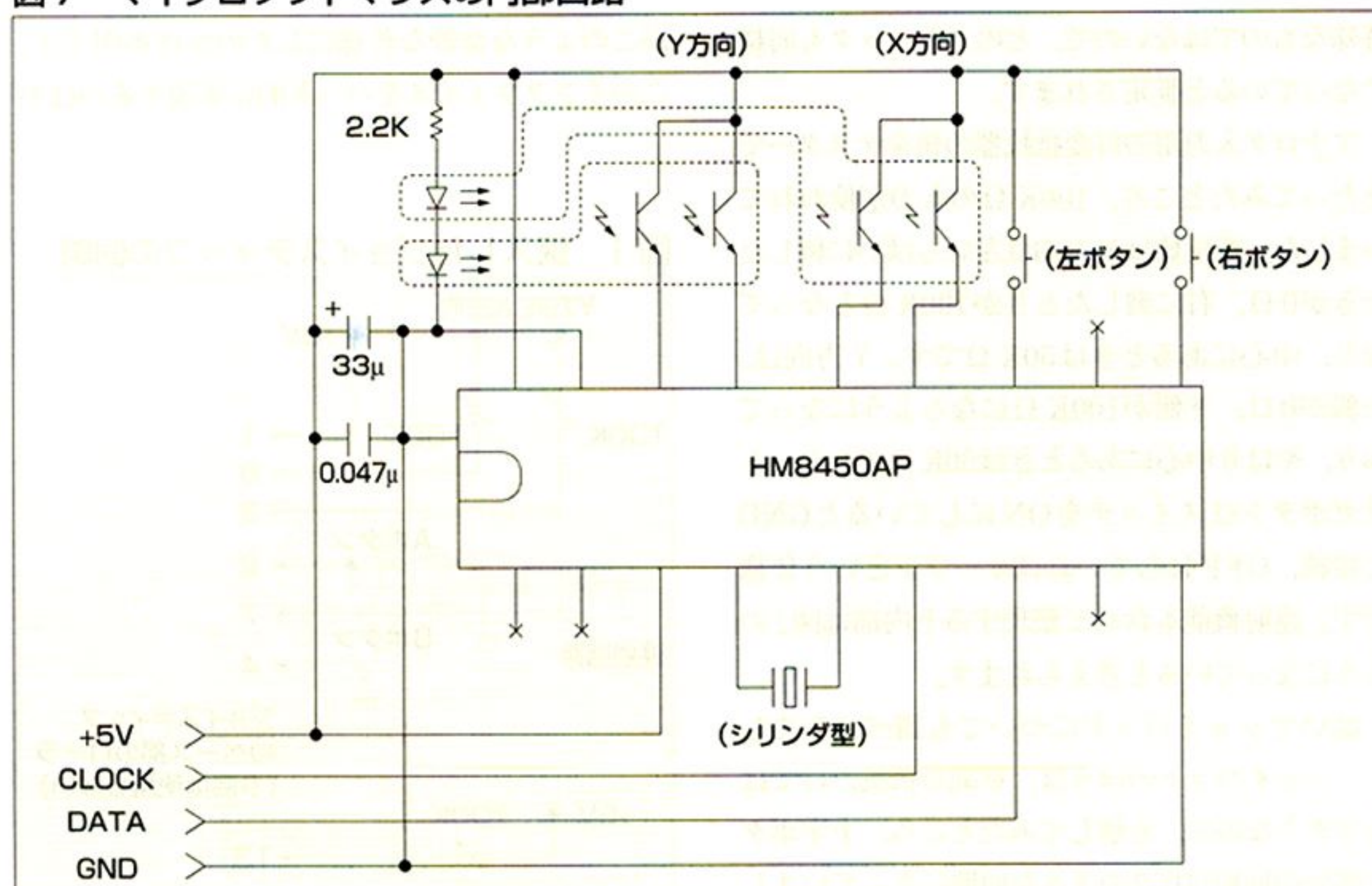
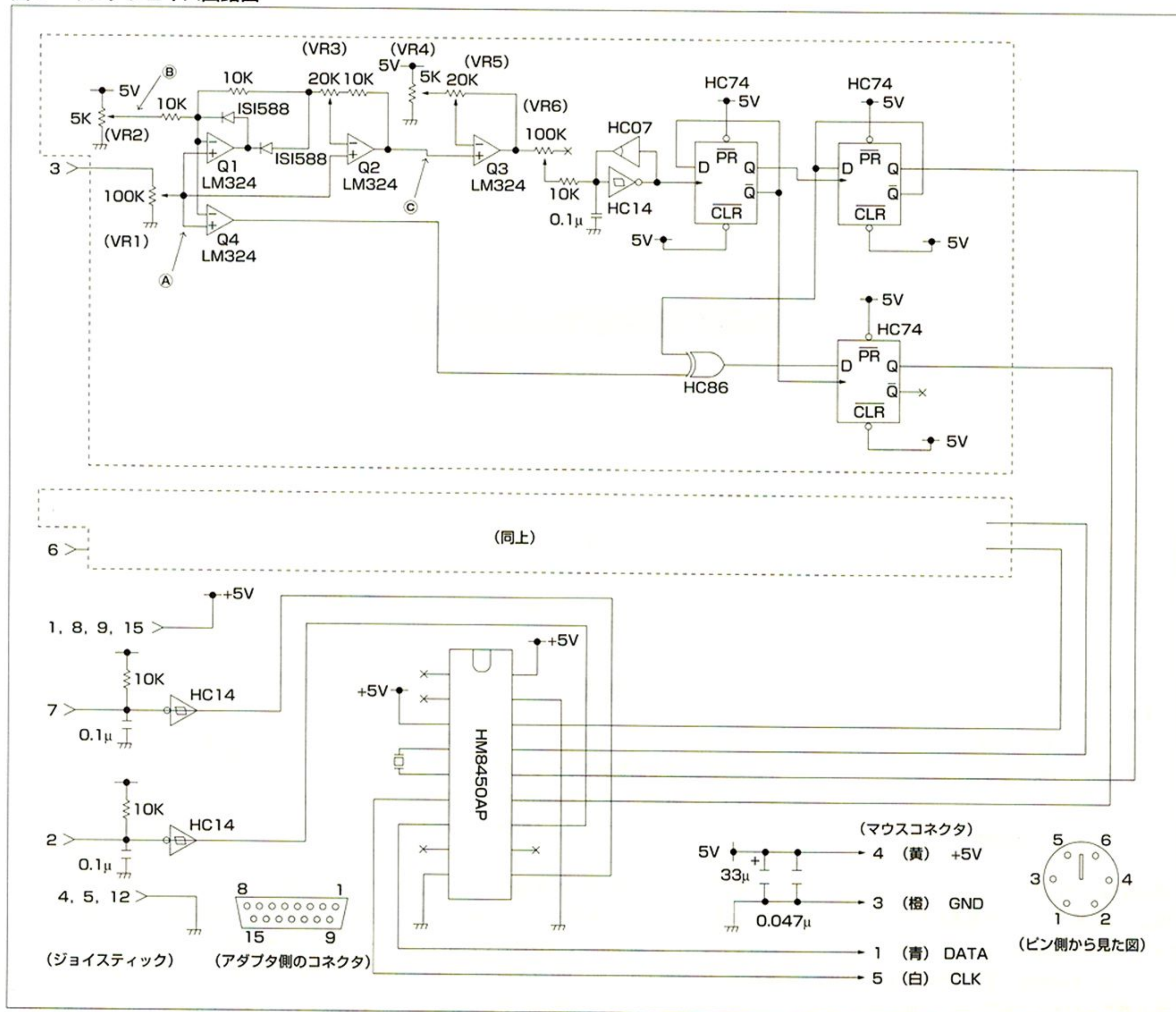



図8　インタフェイス回路図

イスティックのスイッチは押すとGNDと接続される向きですので，ちょうど逆になっているわけです。

変換回路の設計

両方のインタフェイスがだいたいわかりましたので，回路設計に取りかかります。あれこれ試行錯誤してできあがったのが図8のような回路です。オペアンプにはLM324を使用しました。+5V単一電源で動作可能で，多くのメーカーから同等品が出ていますので，入手はそれほど難しくないでしょう。

スイッチは反転させてHM8450APに与えただけです。面倒なのは移動処理です。一度に考えると難しくなりすぎるので，まず次のような特性の発振回路を考えます。

1）ジョイスティックの抵抗値が50KΩ（中央）であれば発振停止
2）50KΩから離れるほど発振周波数が上がる（速く動く）

このような発振出力から1/4周期位相のずれた出力を2つ作ります。さらに，ジョイスティックの抵抗値が50KΩ以上か以下かの判定で，片方の出力を反転させれば目的が達成されるという理屈です。

まず，Q1とQ2の2つで絶対値回路を構成します。スティックがセンター位置にあるときの電圧を基準にして，それよりも電圧が低いとき（抵抗値が大きくなっているとき）には，反転することでスティックが大きく傾くほど電圧が上がるようにするわけです。この出力をさらにQ3に通してレベル合わせをしてHC14とHC07で組んだ発振回路で電圧に応じた周波数で発振させます。Q3によってスティックがセンター位置なら発振せず，傾けると発振開始レベルになるようにしているわけです。

発振回路に使うにはHC14よりも'H'レベル入力電圧が低いHCT14のほうがよいのですが，手持ちがなかったのでHC14で間に合わせました。もう少し凝るならこの部分もオペアンプの発振回路にしておけばHC14のようなばらつきを気にしなくてよかったかと思いますが，今回は簡単にすませました。

発振回路の出力はHC74でフォトインタラプタの出力に似せた波形を得ます。興味のある方はタ

図9-1　ICのピン接続(1)

図9-2　ICのピン接続(2)

表2

品名	型名	数量
PS/2マウス	マイクロソフトマウス	1
DSUBコネクタ	15ピン(メス)	1
CMOS-IC	74HC14	1
	74HC07	1
	74HC86	1
	74HC74	3
OPアンプ	LM324	2
半固定抵抗	5K	4
	20K	4
	100K	4
抵抗	10K	10
セラミックコンデンサ	0.1μF	13
ケース		1
その他	半田,配線材,ネジ等	

イミング図などを書いてみるとよいでしょう。

Q4はスティックがどちらに倒れているかを判定するもので，これがHC86に入力されることで，右向き，左向きの波形となるわけです。

製作

部品表を表2に，主要ICのピン接続を図9-1，図9-2にまとめましたので参考にしてください。まず，マウスの基板からワンチップマイコンと，コンデンサ，水晶振動子を取り外します。半田吸い取り機などがあると楽ですが，片面基板ですから吸い取り線でもなんとかなるでしょう。どのみち基板は使いませんので，面倒なら基板をニッパで割っていくといった乱暴な方法でもかまいません。マウスケーブルも利用しますので，取り外します。ケーブルにコネクタがついていますので，そのまま利用したいところですが，足のピッチがユニバーサル基板とあわないので，直付けにしました。

ワンチップマイコンは念のためソケット付けにしています。18ピンなのですが，18ピンソケットは少々入手しづらいので私は20ピンのソケットで代用しました。1ピン多いので，配線するときにピンを間違えないように注意します。

特に高速な回路でもありませんので，それほど神経を使う必要はないでしょうが，一応IC1個にひとつ程度0.1μFのコンデンサを+5VとGNDの間に入れておきましょう。

調整

実験の繰り返しをしながら組んだこともあり，半固定抵抗による調整個所がかなり多くなってしまいました。調整にはテスタが必要です。電気屋さんやカー用品店などで手に入る安価なものでもかまいません。

調整にとりかかる前に改めて配線チェックをしておきます。特にICの電源ピンはよく確認しておきましょう。ICをソケット付けにしているのであれば，ICを挿入する前にPS/2ケーブルを差し込んでPCの電源を入れて電源電圧チェックをしておきましょう。

間違いがなければいったん電源を切ってICを実装後，再度パソコンを立ち上げてみます。ワンチップマイコンが動作していればマウスが接続されていると認識してくれるはずです。

ここまでうまくいっていれば調整にとりかかります。まず，半固定抵抗はすべて真ん中付近にあわせておきます。次にジョイスティックを接続してVR1を調整して，Q1の+入力(A点)が+2V程度になるように設定します。続いてVR2を調整して，B点も+2V程度になるように調整します。

次にジョイスティックを左いっぱいに傾けてそのときのC点の電圧をメモしておきます。次にジョイスティックを右いっぱいに傾けてそのときのC点の電圧が先ほど記録した値とほぼ同一になるように，VR3を調整してください。

次に発振回路を調整します。シンクロスコープなどがあれば楽ですが，なければPCにつないでマウスカーソルが出る状態にしておきます。VR4とVR5を回すとジョイスティックの傾きに応じてマウスカーソルが移動するようになります。

ここで，ジョイスティックがセンターにあるときにカーソルが動かず，左右に倒したときになるべく速く動くようになるようVR4とVR5を根気よく調整していきます。

最後にVR6でマウスカーソルの最高速度を決定して調整を終わります。Y方向についても同じように調整します。

調整が終わったら適当なアイコンを選択してトリガボタンを引いてみましょう。マウスと同じように使えるはずです。

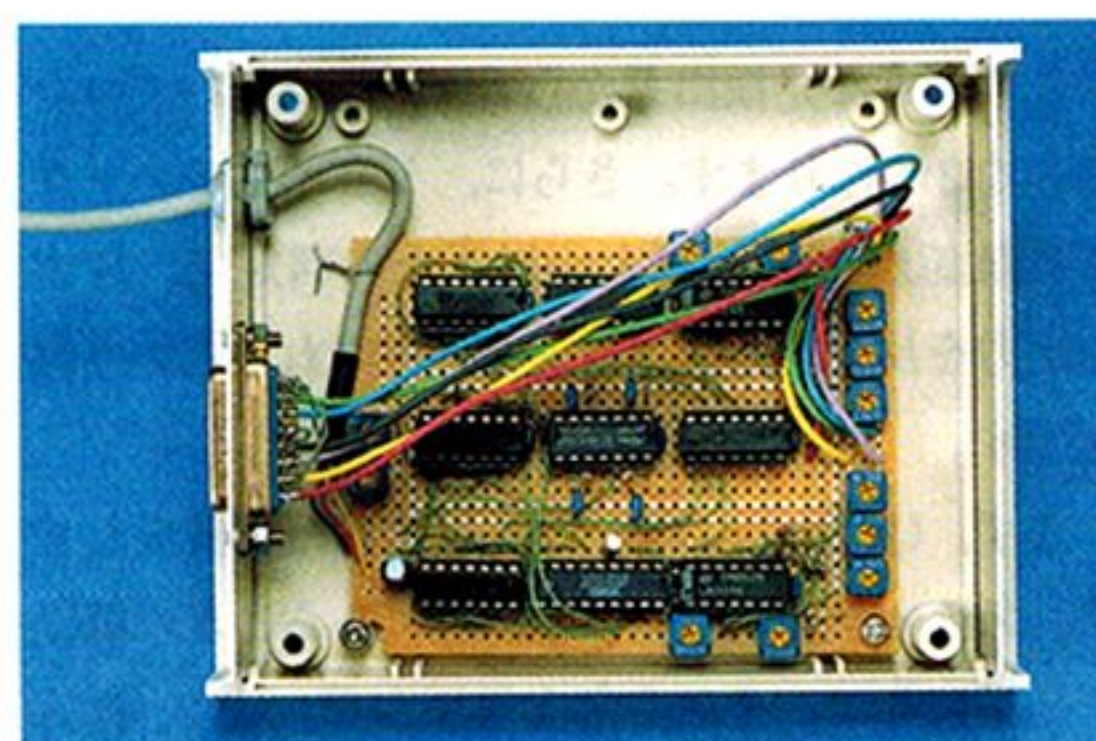
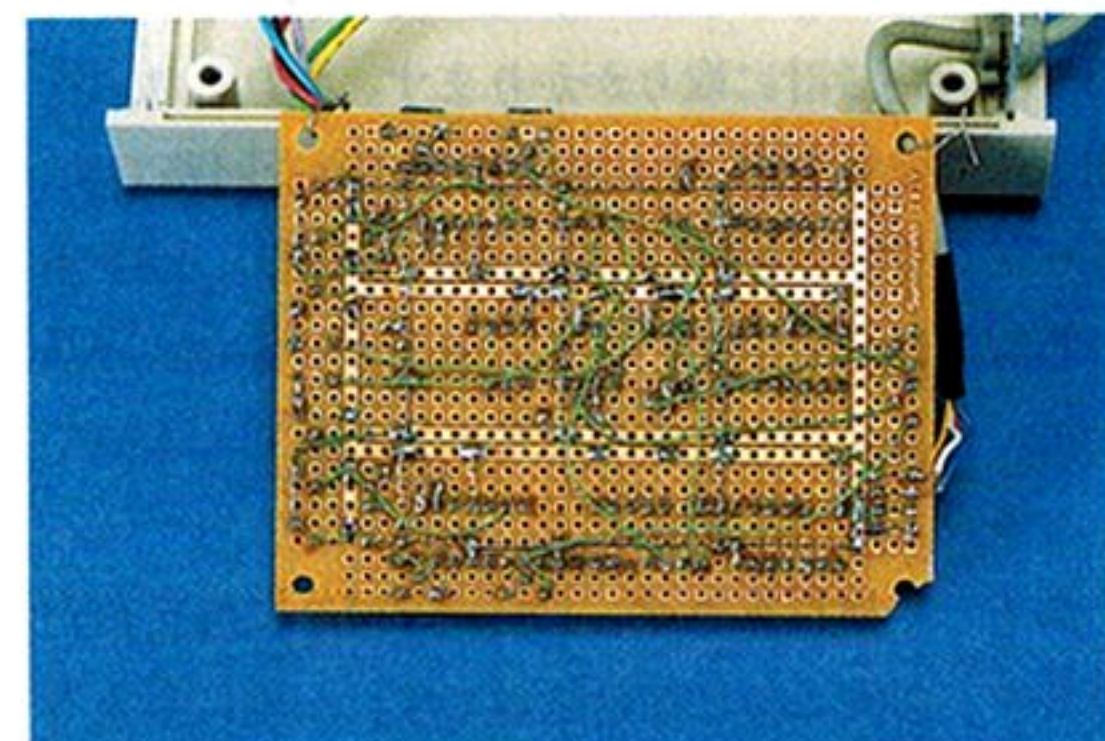
おわりに

とりあえず，ジョイスティックをマウス化することはできました。使った感触では発振回路をもうひと工夫したほうがよさそうですが，とりあえず「ジョイスティックだとこんな感じ」というのは掴めたと思います。案外便利だったのがジョイスティックの連射機能で，連射スイッチを入れてちょっと長めにトリガを引くとダブルクリックとして認識してくれるのです。マウスだと，日に何度もうまく入らないことがありましたが，連射機能だとクリックの間隔は一定ですので，安定してダブルクリックされます。ジョイスティック内部か，あるいはインタフェイス側を改造して連射時に2発しか入らないようにすることも可能でしょう。

では，Oh!Xの再起動を祝して……乾杯！

実際の基板と部品配置

基板裏面

近代PC基本講座

キャッシュメモリ

パソコンを構成するパーツにはさまざまなものがあります。
スペック表などを見ても，その意味するところをちゃんと認識できているでしょうか？
ここではパソコンを構成する基本的な部分を取り上げて
実際の動作などを解説していきます。

桑野雅彦

キャッシュメモリの目的

CPUのバスサイクル時間とメインメモリのアクセス時間の差がそれほど大きくない頃には，CPUに直接DRAMを接続しても，CPUを1クロック待たせる程度でアクセスが可能でした。とくにCPUの性能をぎりぎりまで引き出さなくてはならないような用途を除けば，これで十分CPUの性能向上による恩恵を受けられたのです。

しかし，その後メモリの速度向上をはるかに上回るピッチでCPUの動作クロックが引き上げられるにつれ，メインメモリのアクセス時間がCPU性能向上のネックとなってしまったのです。

インテル系のCPUでも486では33MHz，Pentiumでは66MHz，さらにPenitum IIでは100MHzに達してしまいました。1クロックサイクル時間は，66MHzなら15ns，100MHzでは10nsという値になります。現在メインメモリ用として一般的に使用されているDRAMはアクセス時間がせいぜい60nsや70ns程度ですから，メインメモリをアクセスするたびにCPUは待たされてばかりということになってしまいます。これではいくらCPUの内部動作を高速化しても意味がありません。

いちばん簡単な解決方法はメモリをより高速なものにしてしまうことですが，高速なメモリは値段も高く，メインメモリをすべて置き換えることは現実的でありません。そこで，より現実的な解決方法として考えられたのが，大容量ではあるが低速なメインメモリと，小容量の高速メモリを組み合わせるという方法です。

CPUがメモリをアクセスする様子をよく見ていくと，あらゆるメモリ空間をまんべんなくアクセスするという動きをすることはめったになく，ある狭い領域を集中的にアクセスし，また別のプログラムの実行に移ると，またそのプログラム特有の領域を集中してアクセスするという具合に動くのが普通です。

このようにメモリアクセスに局所性があることに注目して，一度メインメモリから読み出されたデータは高速なメモリに蓄えておいて，次に同じ領域にアクセスしたときにはメインメモリではなく，この高速メモリのデータを渡して済ませてしまおうというのがキャッシュメモリの基本的な考え方です。

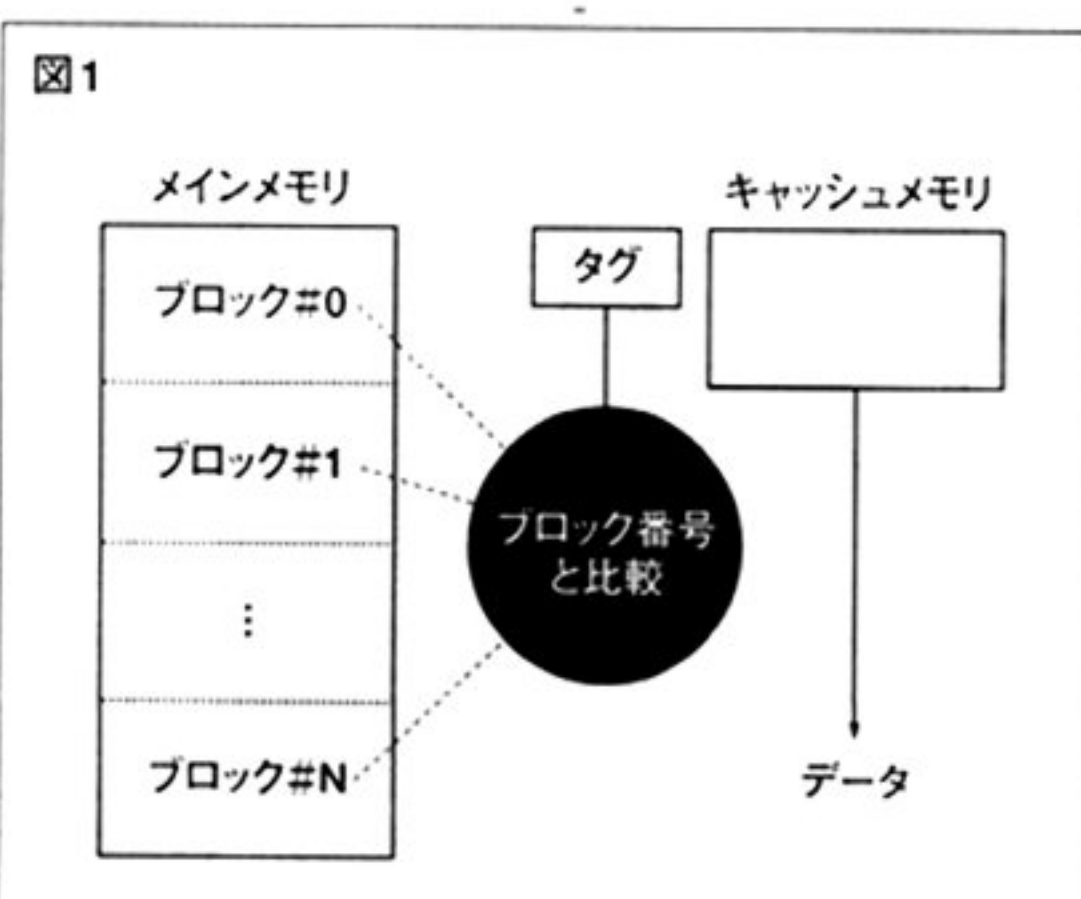
キャッシュメモリの実現方法

キャッシュメモリは，小容量のメモリを大容量のメモリの一部のコピーとして使おうというものです。したがって，キャッシュメモリのある番地の内容がメインメモリのどこに対応しているのかということを管理しておく必要があります。

管理方法にはいろいろな方法が考えられます。とりあえず，話を簡単にするため，ここではキャッシュメモリ容量を256K(アドレス18本)バイト，メインメモリを16M(アドレス24本)バイトとしてみましょう。

❶ダイレクトマップ方式

まず最初に思いつくのは，メインメモリをキャッシュメモリサイズごとのブロックに分割して，キャッシュメモリが現在どのブロックの内容になっているかをレジスタで保持しておくという方法です。先ほどの例でいくなら，メインメモリが16MB，キャッシュメモリが256KBですからメインメモリは64個(16MB/256KB)のブロックに分割されることになります。

これを図にしたのが図1です。この方法は一応キャッシュとしての動作を果たせるように思われますが，メインメモリをキャッシュサイズ分のブロックという，大きなブロックに分割するため，ブロック間をまたいでアクセスをした途端に非キャッシュ空間になってしまうという点には気をつけなくてはなりません。ブロックの外にあるデータを1バイト読んだ途端にキャッシュメモリの内容が丸ごと無効化されてしまうわけです。

これはキャッシュメモリ全体を1個のメモリブロックとして扱ってしまったのが原因です。そこで，キャッシュメモリ自体もいくつかのブロックに分割してしまうという方法が考えられました。この方法はダイレクトマップ方式と呼ばれています。図2はダイレクトマップ方式の考え方を示したものです。

この分割された1つひとつをラインと呼んでいます。先ほどは1個だけだったブロック番号レジスタもラインの数だけ用意します。これをタグRAMと呼びます。

メインメモリとのデータのやり取りはすべてライン単位で行うことになります。1ラインのサイズは任意ですが，あまり小さくするとタグRAMの容量に響きますし，あまり大きくするとキャッシュ内容の更新に時間がかかります。一般的にラインサイズはCPUのデータバス幅の4倍にすることが多いようです。

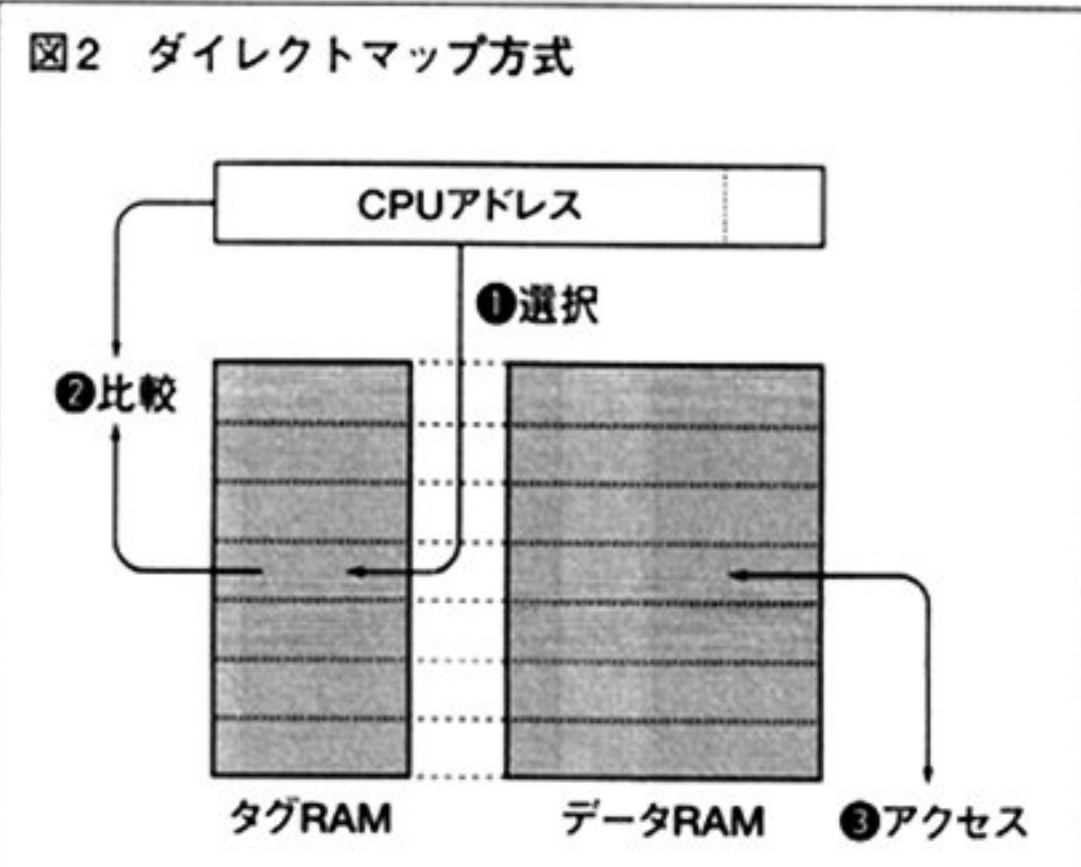
x86ファミリでも486(データバス幅32ビット)のときには1ラインを16バイト，Pentium以降(データバス幅64ビット)は32バイトにしています。ここでは1ラインを32バイトとして考えることにしましょう。

1ラインが32バイトですから，256KBのキャッシュメモリは8K個(8192個)のラインに分けられることになります。タグRAMはそれぞれのラインに対して，上位6ビットのアドレスがどの値なのかという情報を保存しなくてはなりませんので，タグRAMに必要な容量は8K×6ビットとなります。

ここでCPUがメモリリードを行うときを考えましょう。まず，CPUのアドレスがどのラインになるかを判断します。

ここでは1ラインは32バイトですので，アドレ

図1

図2　ダイレクトマップ方式

スのビット0からビット4までは無視し，ビット5からビット17までの13本分を使ってタグRAMにアクセスします。

ここで，タグRAMから出てきた6ビットのアドレス情報と，CPUが出力したアドレス(ビット18からビット23までの6本)を比較して，一致すればキャッシュにデータがあることになりますので，データRAMからデータを引き出してCPUに渡します。タグが一致しなければキャッシュが保持しているのは別のアドレスのデータですので，メインメモリを読み出さなくてはなりません。

このときアクセスされたメインメモリの内容は当然キャッシュメモリに格納しなくてはなりません。もちろん，キャッシュメモリの管理はあくまでもライン単位で行われますので，CPUが要求した1回分だけではなく，1ラインの残りの分のアクセスも引き続き行われることになります。

ダイレクトマップ方式は次に説明するアソシエイティブ方式の連想記憶のような特殊なメモリを使わないで済み，実現しやすい方法といえます。パソコンの場合，CPUの内部キャッシュは2ウェイや4ウェイのセットアソシエイティブ方式が使われますが，L2キャッシュメモリではCypress社のチップセットで2ウェイセットアソシエイティブ方式を採用した例がある程度で，インテルをはじめ大多数のチップセットはダイレクトマップ方式になっています。ただ，AMDのK-6プロセッサの内蔵2次キャッシュは4ウェイセットアソシエイティブの機構が採用されています。

❷アソシエイティブ方式

ダイレクトマップ方式では，それぞれのキャッシュラインが担当するメインメモリ領域が決まっています。メモリアドレスが決まると，それに対応したキャッシュラインがひとつ決まります。このため，同一ラインを使うメモリ領域(下位アドレスがまったく同じ領域)を交互にアクセスされると，キャッシュメモリ容量は十分あるにもかかわらずキャッシュメモリばかりということになります。

これを改善しようというのがアソシエイティブ(連想記憶)方式です。アソシエイティブ方式では，同一のメモリアドレスに対して複数のキャッシュラインが対応可能なように設計します。

キャッシュのヒット率という面で見れば，いちばんよいのはキャッシュのすべてのラインがメインメモリのどのアドレスにも対応できるようにすることです。これをフルアソシエイティブ方式と呼びます。フルアソシエイティブ方式の考え方を図にすると図3のようになります。

フルアソシエイティブ方式では，CPUが出力したアドレスに対して，どのキャッシュラインがヒットしているかを高速に検索しなくてはなりません。このため，アソシエイティブ方式では，通常のメモリとは少し違った構造のメモリが必要です。

通常のメモリはアドレスが与えられると，その番地の内容にアクセスされるわけですが，アソシエイティブ方式で使われるメモリは与えられたデータに一致するものを検索して出力します。このようなメモリを「連想記憶メモリ」と呼んでいます。

フルアソシエイティブ方式では，キャッシュの全ラインを対応可能としなくてはなりません。またアクセス速度を稼ぐためにもかなり高速な動作が要求されます。このような連想記憶メモリを作るのは非常に困難です。このため実際のキャッシュメモリでは，より現実的な方法としてダイレクトマップ方式との折衷案が使われています。

この方法では数個(一般的には2個から8個程度)のラインをペアにし，そのペアの間で連想記憶方式を使います。

たとえば4個のラインをペアにした場合，256KBのキャッシュメモリが64K×4バンク構成になったと考えればよいでしょう。バンク数がN個のものを「Nウェイセットアソシエイティブ方式」と呼びます。2バンクなら2ウェイセットアソシエイティブ，4バンクあるならば4ウェイセットアソシエイティブ方式となります。

図4はセットアソシエイティブ方式を図にしたものです。この例は2ウェイセットアソシエイティブ方式です。

キャッシュを更新するときには，N個のラインからどれかひとつを選び出して行います。この選択方法として一般的なのは，各ラインのアクセスされた順を記録しておき，順番の最後(最近アクセスされていない)のものを置き換えの対象とする方法です。これをLRU(Last Recentry Used)アルゴリズムと呼びます。

❸セクタ方式

セクタ方式もフルアソシエイティブ方式の改良版です。セクタ方式では，複数のラインをセクタという単位にまとめます。連想記憶を使ってセクタ単位で管理するわけです。セクタのサイズをある程度大きくとれば，連想記憶メモリを実現できるようになるという考えです。図5にセクタ方式の考え方を示します。

大きなセクタの内容を丸ごと入れ替えるのは時間がかかりすぎるので，メモリとのやり取りはライン単位で行います。ラインが有効か否かというステータスなどもラインごとに持たせます。

CPUがメモリアクセスをしたとき，まず一致する領域のセクタがあるかが連想記憶で探し出され，一致するものがあった場合には，さらに該当するラインが有効かどうか判定されます。セクタがヒットし，さらにラインが有効ならそのデータがCPUに引き渡されるわけです。ラインが無効な場合にはメインメモリが読み出され，そのデータがキャッシュメモリにもコピーされるのです。

ライトバック/ライトスルー

ここまではCPUがキャッシュメモリからデータをリードするときのことを考えていましたが，メモリへのデータライト動作についても考えておきましょう。

書き込み時のアドレスがキャッシュにヒットしていたとき，キャッシュメモリの内容も更新しなくてはなりません。しかし，キャッシュの内容だけを更新すると，メインメモリの内容との不一致が生じます。これを避けるもっとも簡単な方法はメインメモリとキャッシュの両方に書き込む方法です。この方法をライトスルー方式と呼びます。ライトスルー方式の場合，書き込み動作はメインメモリアクセスと同じ時間がかかります。

ライト時の時間も短縮するためには，ライト時にキャッシュメモリにだけ書き込んで済ませておき，メインメモリに書き込まなくてはならない状況になった時点で書き込んでおくという方法が考

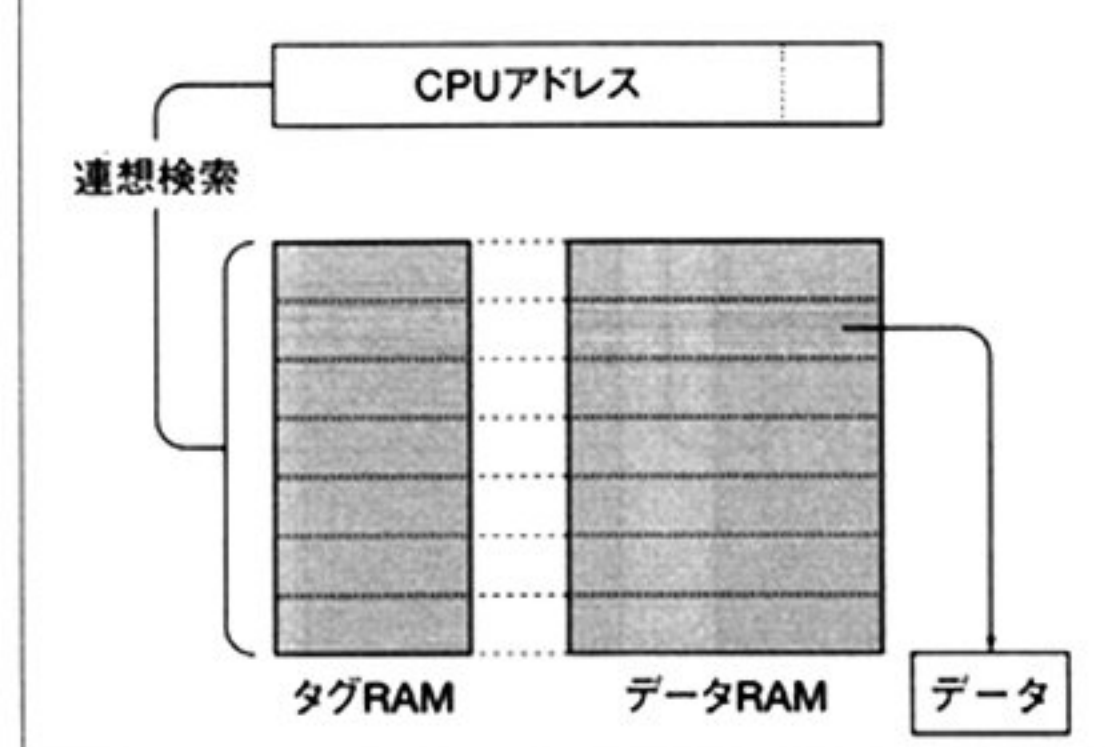
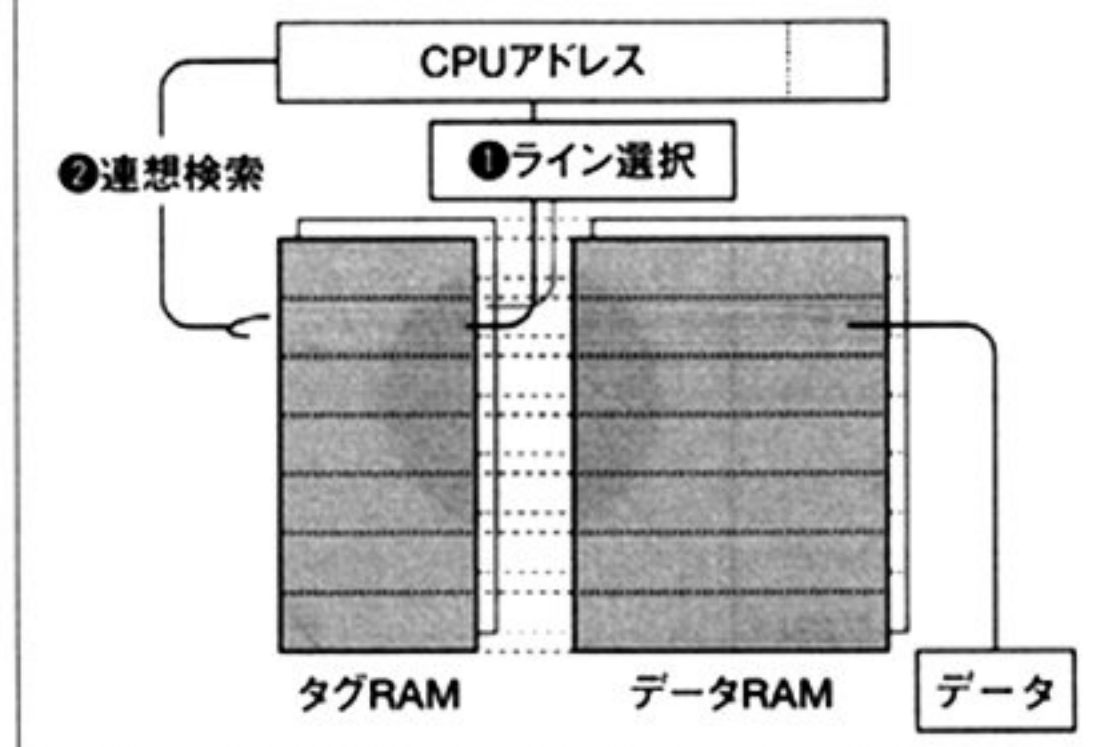
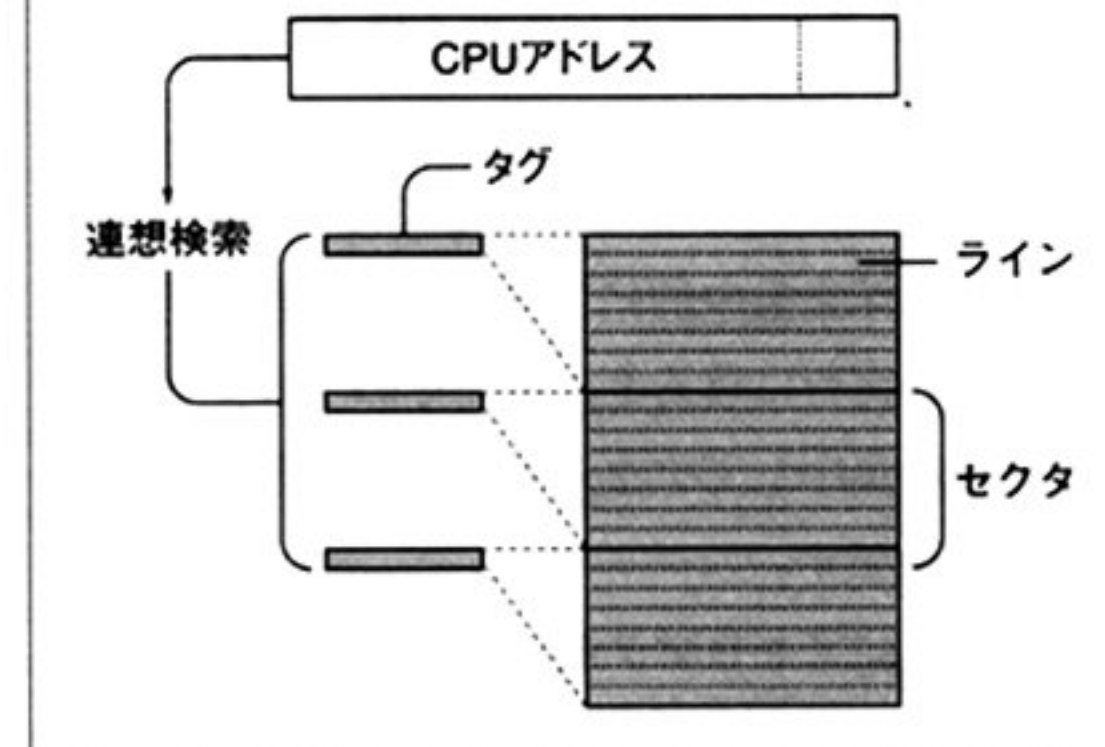
図3　フルアソシエイティブ方式

CPUアドレス
連想検索
タグRAM
データRAM
データ

図4　セットアソシエイティブ方式

CPUアドレス
❶ライン選択
❷連想検索
タグRAM
データRAM
データ

図5　セクタ方式

CPUアドレス
タグ
連想検索
ライン
セクタ

えられます。これをライトバック方式といいます。ライトバック方式の場合，キャッシュメモリの内容だけが更新されていて，メインメモリ上のデータが古いままという状態が生じます。

このため，各ラインに対応してタグビットに加え，キャッシュのみが更新されているかを示すステータス用のビット(ダーティ(Dirty)ビット)が必要になります。

ライトアロケート

通常キャッシュメモリの内容がライン単位で更新されるのは，リード時にキャッシュミスした場合ですが，これをライト時にミスしたときにも更新しようというのがライトアロケートです。

ライトアロケート動作がイネーブルされていると，CPUが書き込み動作を行おうとしているメモリ領域がキャッシュ可能な領域で，しかもキャッシュミスした場合，該当する1ライン分のデータをキャッシュメモリに読み込んでから，書き込み動作を行います。このときのライト動作はライトバックですので，キャッシュだけが更新されます。

単一のライト動作の処理時間を考えれば，直接メモリに書き込んでしまったほうが，キャッシュメモリに1ライン分取り込むよりも速く片付きます。ライトアロケートを行うのは，ライト動作であっても，アクセスがあったということは同一アドレス，あるいは近傍のアドレスへのアクセスがある可能性が高いと考えられるためです。ライトでミスしたときでもキャッシュを更新してしまうことで，2度目3度目のライト動作が高速化できるというわけです。

タグRAMとキャッシュ可能空間

キャッシュの方式を見るとわかるとおり，タグRAMのデータはキャッシュの各ラインに対応し，メモリの上位アドレスと比較されます。したがって，タグRAMはキャッシュのライン数分のアドレスに対応しなくてはいけません。つまり(キャッシュメモリ容量)÷(ラインサイズ)分のアドレスを持っていなくてはならないわけです。

また，キャッシュメモリ容量とタグRAMのデータビット数，およびキャッシュ可能なメモリ空間には次のような関係が成り立つことになります。

(キャッシュ可能なメモリ空間)=(キャッシュメモリ容量)×2^(タグRAMのデータビット数)

たとえば，キャッシュメモリ容量が512KBで，タグRAMのデータビットが8ビットあるとすると，キャッシュ可能なメモリ空間は，$512K \times 2^8 = 512K \times 256 = 128MB$です。つまり，この場合128MB以上のメモリを実装してもキャッシュにヒットしたか否かの判定ができない，すなわちキャッシュできないことになるわけです。

ここで，キャッシュ可能な空間を512MBにしようとすれば，タグRAMはさらに2ビット必要ですので，10ビットを用意しなくてはなりません。さらにライトバック方式を実現するならばダーティビットもタグ情報とともに格納されますので1ビット追加され，合計で11ビット分必要となるわけです。

バースト転送

CPUにキャッシュメモリを内蔵したものではメモリのアクセスの多くが，内部キャッシュメモリとメインメモリの間でライン単位でのやりとりになります。とくにライトバック方式を採用した場合には，読み出しはもちろん，キャッシュにヒットした場合の書き込みも外部バスに行われなくなりますので，キャッシュライン単位でのアクセス頻度が増加することになります。このデータ転送を律義に1回ずつやってももちろんかまわないのですが，CPUの処理性能を落とさないようにするためにはこの転送をなるべく効率よく行うに越したことはありません。

そこで，ライン単位でのデータ転送効率を上げるために考えられたのが，バースト転送モードです。通常のメモリアクセスは，毎回アクセスするたびにアドレスを与えます。しかし，キャッシュメモリとのデータ転送は必ずラインサイズ分まとめて行われ，しかも最初のアドレスがわかれば，2回目以降の転送順序は決まっています。最初のアドレスさえわかれば2回目以降のアドレスはわざわざCPUが与える必要はありません。このようにして行われる転送をバースト転送と呼びます。

バースト転送サイクルで転送されるデータ量，すなわちキャッシュの1ラインのサイズはCPUのバスサイクル4回分にしているのが一般的です。たとえばK6などのSocket7対応CPUではデータバスは64ビット(8バイト)ですので，キャッシュの1ラインは32バイト，486系のCPUではデータバスは32ビット幅ですので，キャッシュの1ラインは16バイトになっています。バースト転送の4回の転送がそれぞれ何クロックで行えるかを3-1-1-1といったような表記で示すこともあります。

CPUの外部に設けられL2キャッシュメモリでも，CPUのバースト転送に対応して，最初のアドレスだけを受け取れば，後続の3回の転送は内部で自動的にアドレスを生成するようにしたものがあります。パソコンL2キャッシュによく使われているPB SRAMもこのようなアドレス生成回路を内蔵したキャッシュ専用SRAMです。

また，最近メインメモリ用としてよく使われるようになってきたシンクロナスDRAMも，DRAMの内部にこのようなアドレス生成回路を設けており，バースト転送が高速に行えるように工夫しています。ただし，シンクロナスDRAMも，内部のメモリアレイの考え方は従来のものと変わりませんので，バースト転送の最初のデータを取り込むまでの時間はさほど高速化されません。

バースト転送時に転送するアドレス順をどのようにするかは，CPUの設計に依存します。世の中のCPUを見ていくと転送順序が必ずアドレス順になる方式(リニアバースト)が一般的なのですが，インテルのPentium等ではアドレス転送順が必ずしもアドレス順にならない，独自の転送順序を採用しています。Intelではこの方式について特許を取得しています。AMDはPentiumと同じ転送方式を使用していますが，Cyrixはこの特許を回避して，リニアバースト方式を採用しています。このため，これらのCPUをインテル製CPUのみに対応したマザーボードに実装した場合には性能が十分に引き出せないことがあります。

インクワイアー(Inquire)サイクル

K6(Pentiumでも)はライトバック方式のキャッシュメモリをCPUに内蔵しています。CPUが外部バスをアクセスするときには内蔵のキャッシュメモリの始末はCPUがつけてくれますが，DMAやPCIバスマスタなどがメインメモリをアクセスするときにはそうはいきません。

CPUの内部キャッシュがダーティ(キャッシュだけが更新されている)なとき，DMAなどがメインメモリを読み出そうとした場合には，キャッシュのデータを渡さなくてはなりません。キャッシュメモリと外部のデータのやりとりはあくまでもライン単位ですので，まず該当する1ライン分のデータをメインメモリに書き出したあとラインを無効(Invalid)状態に設定し，メインメモリのデータを引き渡すという方法をとるよりありません。

また，DMAなどによるメインメモリ書き込みが発生した場合に，キャッシュラインがダーティだった場合にも，やはりいったん1ライン分のデータをメインメモリに書き込み，該当するラインを無効状態にしたあとで，メインメモリのデータを更新するということになります。

このような場合のため，外部からCPUに対して内蔵キャッシュにヒットしているかを問い合わせるためのバス動作をインクワイアー(Inquire)サイクルといいます。

PCIバスマスタなどが頻繁にデータを書き換える可能性のある領域をCPUがたびたびリードするようにすると，キャッシュの更新，インクワイアー，無効化が次々に発生することになり，バスの使用効率が非常に悪くなります。

CPUがチェックしなくてはならないフラグなどはデータ転送用の領域とは別のキャッシュラインになるように配置したり，割り込みで完了通知を行うことで，データ領域を両者が同時にアクセスしないようにするなどの工夫をして，システム性能を向上させることができる場合もあります。

知能機械概論

第100回 「それいっちゃあ，おしまいよ」の人工生命

有田隆也

1987年の9月にアメリカのロスアラモス国立研究所でC. Langtonが開催した人工生命に関するワークショップに刺激されて，人工生命と名づけられた研究領域が各地でスタートしました。そして，今年で11年になろうとしています。人工生命といっても，そのイメージは人さまざまです。なかには，早くも人工生命は終わったなどという声も聞こえてきます。はたして，人工生命とはいったい何者か？ 10代になったばかりというこの人工生命は，すでに死んでしまったのか？ あるいは永遠の命を手に入れるのか？ のんびりと考えてみます。

冷蔵庫の５分の１

人工生命ったって，そんな言葉は初めて聞いた，とおっしゃる方だって，当然おられるでしょう。そういわれれば，そんなに知られた言葉でもないような気もしてきます。では，試しに，インターネットの検索エンジンgooで，どのくらいのページがひっかかるか調べてみましょう(8月13日現在)。

「人工生命」という言葉で日本語のページを探索すると2374ページがヒットしました(まあ，そのうちの1％ぐらいは，僕の作ったページがカウントされているような気もしますけど)。といっても，このページ数が多いのか少ないのかよくわからないといえば，わかりませんね。じゃあ，「冷蔵庫」ではどうでしょうか？ これだと25758です。やはり，すごいですね。いや，冷蔵庫もすごいけれど，冷蔵庫の10分の1もあるのですよ，10分の1も。

今度は，英語でつづられた場合も含めて，人工生命 OR "Artificial Life" OR ALIFEで，日本語で書かれたページをブーリアン検索してみると，先ほどの倍の5697ページにまで達しました(英語だけのページも一部含んでしまうようですけど)。これで，冷蔵庫の5分の1というわけです。ちなみに，これでは，冷蔵庫にちょっと不利なので，こちらも，冷蔵庫 OR 電気冷蔵庫 OR refrigeratorで，ブーリアン検索しても，27556に増えただけです。やはり，冷蔵庫の5分の1というところですね。

学会での歓迎度合

研究としての認知度ということをいうならば，やはり「学会」様という権威方面を見てみることにしましょう。人工生命という言葉が認められているかとか，人工生命を主テーマにした論文が掲載されているかということです。答えからいいますと，国内の学会でもこれが意外と認知されているのですよ。

私自身も，たとえば，情報処理学会という学会の論文誌に，すでに3年前に「自己組織系集団による通信の進化の試み」なる論文を，また，去年にも「捕食者・被食者間相互作用に基づくカラーパターン進化の解析」なる論文を載せていただきました。あ～，ありがたし。

電子情報通信学会というこれまた堅そうな名前を持つ学会の最新号に「論文誌専門分野分類表について」という記事があります。論文を投稿するときに，この学会ではその論文がどういうジャンルに属するかということを表す分類表のコードを一緒に添付するということになっているのですが，その分類表が改正されたという記事です。さて，それを見てみると，あるのですよ，「人工生命」が。しかも，迷いそうなほどに。たとえば(/は，下の階層に移ることを意味します)，

・情報・システム/情報処理/人工知能，認知科学/遺伝的アルゴリズム，人工生命

というジャンルもあれば，

・情報・システム/パターン処理/バイオサバネティックス，ニューロコンピューティング/人工生命

という分類もあるのです(関係ありませんが，情報処理とパターン処理という二分化がややや気になりますけど)。これだけにとどまらず，ほかにも，「創発」だとか，「複雑系」だとか，人工生命のキーワードを持つ分類がいくつかあるのです。

人工生命に関連しそうな国内の学会は，むろん情報処理学会や電子情報通信学会だけではありません。人工知能学会とか，計測自動制御学会だとか，ほかにもさまざまな学会がむしろ好意的に受け入れてくれているといってよいのではないでしょうか？ まあ，これはおまえの個人的な感触だろうといわれればそうですが。

いうまでもなく，英文で投稿するならばさまざまな機会があります。Langton自身が編集するArtificial Life (MIT Press発行)というそのものずばりの名前がついたジャーナルもありますし，国際会議もいくつもあります。

「教祖」と呼ばれる人

人工生命という研究領域を開拓した最大の功労者は，なんといっても「教祖」Langtonです。彼は1987年のワークショップを開催して以来，マスメディアにも繰り返し登場し，人工生命の研究意義を説いています。そのワークショップも最初は「生物システムのシンセシスとシミュレーションに関する学際ワークショップ」なる正式名称を持っていましたが，人工生命という言葉が認知されてきたので，現在では「人工生命に関する国際会議」という名前に変わっています。

図1　1次元セルオートマトンの動作例

時間の経過 ↓

0	0	1	1	0	1	1	0	1	0	0	1	0	1	1	1	0
1	1	0	0	0	0	0	0	1	1	1	1	0	0	1	0	1
2	0	1	0	0	0	0	1	0	1	1	0	1	1	1	0	0
3	1	1	1	0	0	1	1	0	0	0	0	0	1	0	1	0
4	0	1	0	1	1	0	0	1	0	0	0	1	1	0	1	0
5	1	1	0	0	0	1	1	1	1	0	1	0	0	0	1	1
6	1	0	1	0	1	0	1	1	0	0	1	1	0	1	0	1
7	0	0	1	0	1	0	0	0	1	1	0	0	0	1	0	0
8	0	1	1	0	1	1	0	1	0	0	1	0	1	1	1	0
9	1	0	0	0	0	0	0	1	1	1	1	0	0	1	0	1
⋮							⋮									

図2　自己増殖ループの初期パターン

```
  2 2 2 2 2 2 2 2
2 1 7 0 1 4 0 1 4 2
2 0 2 2 2 2 2 2 0 2
2 7 2         2 1 2
2 1 2         2 1 2
2 0 2         2 1 2
2 7 2         2 1 2
2 1 2 2 2 2 2 2 1 2 2 2 2 2
2 0 7 1 0 7 1 0 7 1 1 1 1 1
  2 2 2 2 2 2 2 2 2 2 2 2 2 1
```

図3　増殖の様子

写真：Alife Ⅵで配られた人工生命グッズ

その6回目の会議がこの6月にロサンゼルスで開かれました（文献1参照，写真1：Alife VIのロゴ入りサングラスまで配られてしまうところがすごい）。そこでLangtonが10分以上話すのを聞くチャンスが2回ありました。ひとつは，国際会議初日のワークショップで，Swarmという人工生命の研究環境に関しての概論で，もうひとつは国際会議最終日の閉会の言葉でした（正確には，もう1回，ATRの土佐さんのインターラクティブシネマのデモ会場で感情たっぷりにスクリーン上のCGの女性に語りかけていました）。基本的にいっていることは前からずっと変わらないなあという感じ（教祖は宗旨替えしません！）でして，人工生命研究の意義について明快に語っていました。

少なくとも「人工生命」という言葉が認知されてきたひとつの要因として，カリスマたる彼自身のふりまくイメージについては見落とすことができないと思います。彼のアウトサイダー的な経歴，ハンググライダー事故による九死に一生を得たという大事故，そして長期入院中にひらめいたという彼の頭の中における抽象的な生命パターンから人工生命のひらめき，……。実際，このような「伝説」は，人工生命という言葉自身の持つSF的な味わいとともに広まっていき，日本では，お祭り騒ぎだ，流行だと盛り上がってきたのは事実でしょう，良かれ悪しかれ。

セルオートマトンの基本

人工生命という研究領域の教祖，あるいは伝道者として，Langtonは中心的な仕事をしてきました。それは誰もが認めるところです。でも，彼の具体的な研究内容に関しては，もしかしたらあまり知られていないかもしれません。そこで，せっかくですから，セルオートマトンに関する彼の研究をさらっと紹介することにします。

まず，セルオートマトンについてごく簡単に述べておくことにします。John von Neumann（現在の計算機の原形を考えた天才）がS.Ulamと考え出したセルオートマトンというのは，次のようなものです。

まず，セルが並んでいるn次元平面上（nは1か2が普通）を考えます。各セルはそれぞれ状態をとります。その状態はルールに従って変わっていきます。ルールは着目しているセルの状態とそのセルの近傍のセルの状態に応じて，そのセルが次のステップにどの状態に変わるかを決めるものです。ルールは全セルに対して同時に適用されます。

1次元セルオートマトンの例をひとつだけ示しておきましょう。各セルは0か1かの2つの状態のいずれかをとり，次のステップでどの状態になるかは，現在の自分，および，自分の真横の2つのセル（3近傍）の状態によって決まるものとします。したがって，3つのセルのとる状態の組み合わせのそれぞれに対して遷移後の状態がルールで定められることになります。たとえば，次のようなルールがあったとします。

111	110	101	100	011	010	001	000
↓	↓	↓	↓	↓	↓	↓	↓
1	0	0	1	0	0	1	0

ここで，たとえば2つめの遷移は，左のセルの状態が1で，右のセルの状態が0で自セルの状態が1の場合，そのセルは次のステップで0になることを表しています。このように全組み合わせ8個の遷移を定義することによって，ひとつのルールが表されます。

このルールに従ってどのようにパターンが遷移していくか，試しに見てみましょう。図1は，時刻0から時刻9までの状態が遷移していく様子を表しています。初期状態はでたらめな0と

1の並びです。端のところは右と左がつながって筒のようになっていると考えてください。この例では，時刻7まではばらばらなパターンを生み出しますが，突然，時刻8になってから時刻0と同じ状態になり，それ以下も同じように周期8で繰り返すことがわかります。

これは1次元のセルオートマトンの例ですが，2次元の場合には，隣接するセルの現在の状態から各セルの次の時刻における状態が決まり，2次元平面上のパターンが刻々と変化していくことになります。有名なライフゲームも2次元セルオートマトンの一種です。

セルオートマトン上でなんと普通のコンピュータが作れてしまう話だとか，挙動の分類の話だとか，生命との関連とか，興味深い話がたくさんあります。なにが面白いのかといわれれば，要するに，ルール自体は簡単なものなのだが，セルの局所的な状態遷移がほかの場所に影響するのに時間がかかるので，それらの影響の総和としてのパターンが予測不可能な複雑な挙動を示して面白くて，しかもその現象というものが生命の本質と関連がありそうだというのです。

自己増殖するループ

さて，Langtonの行ったセルオートマトンに関する意義ある仕事として，「自己複製ループ」を挙げましょう(文献2)。彼は頭をしぼって8状態5近傍の2次元セルオートマトンを考え出しました。遷移の数は数百にのぼります。最初のパターンは，図2のような形をしているのですが，状態遷移を開始すると，飛び出た部分が少し伸びたあとに左に折れ曲がるということを3回繰り返します。そうすると，連結した四角形が2つになり，ついには，その2つの四角形の間が切れてしまうということになります。そして，また，それぞれの四角形が腕を伸ばし始めます。これでひとつが2つに増殖したというわけです。これを繰り返していきます。このようにして，元のパターンが自己増殖を繰り返していくというわけです。

この説明を読んで，あるいは，このセルオートマトンの動きを見て(フリーウェアがあるので興味のある方は入手してください)，これがいったいどういう意義を持つのか？ と不思議に思う方も少なからずおられるかもしれません。この自己増殖ループの持つ意味を知るには，von Neumannという天才の行った仕事，そもそもなぜセルオートマトンなるものを考え出したかということを知らねばならないのです。しかし，そこらあたりから話をすると，ずいぶんと長～い話になりますので，思いっきりはしょって，次のようにいいきってしまいましょう。

von Neumannは，生命の本質は計算過程であるとの発想から，それを示すために，生命現象のうちの特に重要な自己増殖性に着目し，それをなんとか機械でもできることを証明しようとしました。そのために，セルオートマトン上で，万能チューリングマシンが構成できることを示し，さらに，それを利用して自己増殖マシンを構成できることを示したのです(実際に動かそうとするととても複雑なので，シミュレーションすら最近までできませんでした)。

さて，Langtonの自己増殖ループの背景には，次のような彼の主張があります。「確かに計算万能性は自己増殖マシンを作るための十分条件ではあるが，必要条件ではないのだ(＝万能チューリングマシンがあればそれを利用して自己増殖マシンはできるが，なにも万能マシンなる複雑なものをわざわざ使わなくてもできるのだ，生物だって万能マシンのようには見えないし)」ということで，万能チューリングマシンを構成するということをスタートに持ってくるのはやめて，比較的簡単な自己増殖そのものを実現するずっとシンプルなモデルを作り上げたというわけです。

ここでもっとも問題になるのは，自己増殖性といったって，単にパターンが増えればいいというならば，たとえば，早い話，1次元セルオートマトンで，初期状態として，

00001000

であったパターンが，少し時間が経過したら，

00101010

になったならば，これだって，自己増殖したのだという主張がありえてしまうかもしれないということです。これも自己増殖というならば，自己増殖なんて，生命の本質とはまったく関連のなさそうな全然面白くないものになってしまいます。

Langtonは，上記のような取るに足らない例を排除するために，「状態遷移ルールばかりに依存して増殖を行うものは排除すべきであり，パターンの構造自体に主に依存して増殖が行われるべきものが生物システムの研究にとって有意義なのだ」と主張しています。

ちなみに，Langtonの行ったこととしては，自己増殖ループ以外にも，いわゆるLangtonの蟻，それからλパラメータの概念などがあり，これらもセルオートマトンに関連するものです。最近，特に力を入れているのは，人工生命や複雑系的なアプローチのための総合的なソフトウェア環境であるSwarm(実際，Langtonは最近Santa Fe研究所をやめて，自身が設立したSwarm Corporationに移りました)などがあります。

あるいは50歳の人工生命

天才von Neumannが"General and Logical Theory of Automata"というタイトルで先に触れた自己増殖マシンのアイデアを講演してから，今年でちょうど50年になります。人工生命の概念を唱えたLangtonの自己増殖ループ，あるいは彼のほかのいくつかの研究も，このNeumannが提起した問題意識に源を持つひとつの流れの上に載るものです。人工生命研究はさまざまな領域に広がっていますが(たとえば僕が行っている言語の進化など)，生命の普遍的な性質を情報という枠組みの中で計算モデルとしてとらえようとする試みは，人工生命研究の核を構成しているといえます。

このような方向性を持つさまざまな研究が人工生命という名の下でなされてきました。実際，先ほど開かれたAlife VIでも，佐山弘樹さんはLangtonの自己増殖ループに死の概念を導入した興味深いモデルを発表しました。死というものを新たなパターンを作り出すための空間を作り出す現象として積極的にとらえ，Langtonのモデルのように，中心から珊瑚礁のようにパターンが固まっていってしまわずに，よりアクティブで複雑な振る舞いをさせることに成功したのです。

今回の人工生命国際会議の動向を見ると，一面，分子レベル，細胞レベルでの計算生物学的な傾向が強まって，発生，細胞分化などのメカニズムを解明する生物学的アプローチの方向に人工生命の大きな流れができつつあることが感じられます。しかし，一方で，佐山さんの研究のように，実際の生物から一定の距離を置いて束縛や地上における偶然性から逃れた抽象的な計算モデルの領域で，生命とはなにかということを追究するアプローチが世界中でなされてきています。

「これが人工生命です」

最近，人工生命に関するテレビ番組をNHKで2つ見ました。ひとつは，「人工生命への挑戦」(これはBS放送)で，もうひとつは，「人工生命が生命の謎に迫る」です。前者はフランスで番組が作られたようですが，インタビューや映像素材などの中には同一に見えるものが少なくありませんでした。同一の企画というか取材ソースから2つ番組を作ったのかもしれません。

ですから，2つの番組を見比べて，日本とフランスの国の違いが伝わってきて，興味深いものがありました。フランス版では，人工生命の持つ哲学的な問題などが指摘され，また，全体の雰囲気がアートっぽかったのに対して，日本版は，ロボットに関する研究事例の紹介にかなりの時間を使い，また，丁寧に人工生命の概念を説明しようとする姿勢が感じられました。

国の差は横に置いておくとしまして，日本版のほうを見ていて，奇異に感じたのは，ソニーが開発したペット型ロボットが映されている映像に対していきなり「人工生命」と文字が出たり，Tierraモデルの画面でプログラムひとつを表す青線に対しても「人工生命」と説明するところでした。また，T. Rayのインタビュー場面で，Rayがプログラムの寄生などについて説明している場面で，「青いプログラムが～」と英語でいっているのに，(無理やり)「青い人工生命は」，などと翻訳音声を上からかぶせているところにも違和感を覚えました。

要するに，「人工生命」=「生きているみたいに動くロボット，または，生きているみたいなパターンやグラフィック」というような定義を前提としているのです。こういう把握は，「見栄えが第一」的なテレビの宿命によるところもあるのでしょう。でも，それ抜きでも，もともと人工生命研究というものはそのようにとらえられがちであることは私自身ずっと感じてきたことでした。

「このロボットが人工生命です」といっちゃあおしまいよ，という感じです。生命なんてこの程度のものだと自分で宣言しているようなものですし(ちなみに，そのロボットがつまらないといってるのではまったくありませんので)。

とにかく，人工生命研究の究極的な目標(生命を構成的に研究することにより，生命とはなにかということを究める)がすでに到達されているようなものですからとにかくヘンです。また，単にコンピュータグラフィックを使って，生きているみたいな見かけを作るということが，人工生命研究の主なテーマのように受け取られる点(重要な一分野には違いありませんが)でもまずいと思います。さらに，人間型，あるいは生物型ロボットの研究が人工生命の研究の主テーマならば，「人工生命」という言葉は単にロボティクスの一分野に名づけられた(魅力的でもあるが，そのうち忘れ去られそうでもある)名前のように思われてしまいます。

人工生命の全体像

私の研究室(通称：人工生命ラボラトリ)にこの4月入ってきた修士1年の院生がどこかのBBS上で「今度，人工生命の研究をしている研究室に入る」というようなことをいったところ，「まだ，人工生命の研究をやっているところがあるのか？」的な反応が返ってきたと聞きました。

(ふ～っ！)と私は思いました。でも，(やっぱり)という気もします。人工生命が，ロボット研究の一部に対してつけられた流行中の言葉だとか，生き物風の動きをするコンピュータグラフィックにつけられたキャッチーな名前だとかのような理解のされ方ならば，それは，単なる流行かもしれませんし，遅かれ，早かれ，終わる宿命があるでしょう。また，なんらかの応用の分野で成果を出さなければ意味がないような工学的なものだけならば，実用になる部分だけ利用されてぽいっと捨てられるという感じになってしまうかもしれません。

Langtonに関する伝説のほうではなく，彼の行った研究のほうに注目すると，人工生命が11歳になったばかりのほやほやではなく，普遍的な生命性の問題にアプローチし続けているvon Neumannからの研究の流れを引き継ぐ50歳の研究領域であり，しかも，さまざまな成果を残してきたことに気がつくと思います。

むろん，人工生命とひと口にいっても，さまざまな流れがありますし，とらえ方があります。従来のさまざまな分野においても，いってみれば単なる手法としてですが，さまざまな形で根づき始めています。

ただし，従来の学問領域，たとえば，生物学にも，工学にも，情報科学にも，どこにも位置づけることができないような部分，そこにこそ，人工生命研究の核心領域があり，そこから，さまざまな研究が流れ出しているというとらえ方をするのが，人工生命の全体を理解するいちばん素直な方法ではないかと考えるのです。

Illustration：Haruhisa Yamada

参考文献)

1) C. Adami, R. K. Belew, H. Kitano and C. E. Taylor, eds , Artificial Life VI: Proceedings of the Sixth International Conference on Artificial Life , 1998, MIT Press.

2) C. G. Langton, "Self-reproduction in Cellular Automata", Physica, 10D, pp. 135-144, 1984.

参考ホームページ)

検索エンジンgoo：定番サーチエンジン
http://www.goo.ne.jp/

Dr. Christopher G. Langton：
C. Langtonの現在のホームページ
http://www.trail.com/ cgl/

Swarm Corporation：Langtonの作った会社
http://www.swarm.com/

Structurally Dissolvable Self-Reproducing Loop：佐山さんの
「死の概念を持つ自己増殖ループ」
http://proton.is.s.u-tokyo.ac.jp/ sayama/sdsr/

人工生命の宝庫：
私の作った人工生命に関する解説付きリンク集(フリーウェアが充実
http://www2.create.human.nagoya-u.ac.jp/ ari/stuff/alifesoft.html

10歳の人工生命：
人工生命研究一般と私の研究の説明
http://www2.create.human.nagoya-u.ac.jp/ ari/stuff/10years.html

さらば過渡期といおう

小笠原陽介

パソコンを取り巻く結界とそれを生み出す呪文

パソコンという厄介なシロモノに関わってずいぶん経つのだが，この世界の不可思議さについては，いまだに納得できないことが少なくない。

たとえば，ほとんどすべてのパソコンが，買って3カ月から半年もすれば旧機種となり，型落ち品として値段がガクンと安くなること。もちろん，どんな世界でも似たようなことはあるのだが，それにしても，よほどタイミングが悪くなければ，まぁ半年や1年は新機種だぜフフフ，と喜んでいられるものだし，たとえ後続が出たところで，いきなり旧機種が半額近くで叩き売り状態になったりはしないだろう。

あるいは，いったん買ったパソコンも，2年もすれば機能や性能を補うなにかの買い足しが必要不可欠になり，4年もすればそういった手を施してさえ使いものにならないかのようにいわれてしまうこと。使い減りが激しい種類の機械であればともかく，パソコンの中で消耗を伴う部品はさほど多くない。いや，そうした消耗系のパーツにすら物理的な不具合がなにひとつないまま，数々のマシンがポンコツ同然に扱われるようになっていくのだ。

もちろん，最新の技術をフルに利用したいなら，それはそれでよい。3Dアクセラレーションを目一杯使って，これまで見たこともなかったようなゲーム世界に遊んでみたい，そのための投資はいとわない，というのは立派な目的意識だと思う。だが，メールのやり取りやWebページを見て回るくらいのことしかしないはずの人が，DX4/100MHzクラスのマシンを「使いものにならない」と罵り，CPUやビデオカードを買い替えてはベンチマークテストの値を見て喜んでいるというのは，少なくとも私には奇異なことに見えてしかたがない。

さらにまた，パソコンをなにがしかの用途に役立てるためのソフトウェアに，1年に1回ほどの割り合いでバージョンアップという名の変更が加えられ，操作方法が変わったりファイル形式が変わったりして，ユーザーが絶えず新しいルールに従わされることにも納得できていない。いや，その程度ならまだしも，2～3年に1回はOS自体の変更が行われ，より根本的なルールまでもが，ほぼ面目を一新してしまう。使い古されたたとえになるが，いわば自動車のアクセルとブレーキの位置を移動するようなことが，実に頻繁に起きているのだ。

たとえばDOSからWindows，あるいはまたWindows 3.1からWindows95といったように，それによって「できること」が格段に増えたり，また同じ事柄を達成するまでの手順が大幅に簡単になったりするのであれば，ルールの変更に従ってでも，新たなメリットを取る，という選択は意義がある。だが，せいぜいワードパッドでもあれば足りるような文書しか作らない人が，ワープロソフトのバージョンアップのたびに律義につきあってインストールするのは，私の目には真に奇妙なことに映る。

どうしてこのような不可思議なことが起きているのかを，長年にわたって考えてきて，いまもまだ「これだ！」という答えにはたどり着けずにいる。だが，部分的にはいくつかわかってきたことがある。ひとつは，どうもパソコン界には強力な結界が張られているらしく，世間のほかの部分で通用している常識の類いが，ほとんど適用されないようになっているということだ。

もうひとつは，そのような結界を維持するために，パソコン界には，いろいろな呪文が飛び交っているということだ。そして，そのなかでも，私がもっとも強力だと考えている呪文は，

「いまはパソコンにとって過渡期である」

というものだ。

終わりなき過渡期のなかで変わりなき不満と焦燥

パソコンは過渡期のものだから，もう少し落ち着くまでは，とにかく時代の流れに遅れてはいけない。誰がいうともなく，それはほとんど真理であるかのように信じられてきた。

新しい潮流，新しい情報，新しいマシンに新しいソフト。それらをいち早く手に入れ，噛み砕き，自らの血肉としていくことこそが，この過渡期を乗り切る唯一の術なのだ，と。

ハードウェアにせよソフトウェアにせよ，次々に登場する新製品は，どれも非常に魅力的に見える。「概念を一新する新機軸」や「圧倒的な高機能」，加えて「快適な動作を約束する高速性」に「常識破りのコストパフォーマンス」などなど，華やかな売り文句に引きつけられてしまう。たとえその内容が，屋上屋を重ねるような仕様追加の結果であろうと，トリッキーな動作をしていようと，旧バージョンより重くなろうと，新機能を使うあてがなかろうとも，やっぱり新しいもののほうがよいと思えてくるものだ。

だが，ふと立ち止まって考えたとき，大きな疑問にとらわれる。そうやってパソコンを追い続けて何年になるのだろう，と。そうやって常に新しいものを求め続けてきて，いったいなにが変わっ

たのだろうかと。

確かに，ワンボードマイコンの時代から約20年，かつてのスーパーコンピュータ並みのマシンパワーが個人の机の上に載ってしまうまでになったことは，大きな変化には違いない。だが，そうやって「新しいもの」を追いかけながら使ってきたユーザー自身は，どれだけ変わったのだろう。新しいマシンに次々に乗り換え，追加パーツを買いまくり，書籍雑誌を読みあさりながら，その実，いつも不満と焦燥に駆り立てられていただけではなかったのか。

よりよいマシンや，よりよい環境を求めること自体は，本来，決して悪いことではない。よい仕事をするためによい道具を求めるのは，ごく自然なことだ。だが「過渡期」という呪文は，ユーザーにこうした「必要→よい道具」という目的意識を忘れさせ，必然性なく最新機種や最新バージョンに執着させる働きがあるのだ。

性能にさほど遜色のない旧モデルが叩き売りになるのも，まだ使いようのある486ベースのマシンがゴミ扱いされるのも，この呪文に縛られているせいに違いあるまい。

最新機種や最新バージョンを手に入れること自体が自己目的化した途端に，あらゆる現実は不満の対象でしかなくなってしまう。マシンの遅さを嘆き，ソフトウェアの使い勝手を罵り，高価なものが手に入らない不遇をかこつ。だが，そんな風にいろいろなものを呪いながらパソコンを使うのは，不幸なことではないだろうか。もうそろそろ，この「過渡期」という呪文の束縛から自由になってよいと思うのだ。

時間の流れとともに技術改良が進んでいる以上，パソコンは今後ともどんどん進歩していくだろう。前年比50%増，100%増といった数字も，引き続き見られるのかもしれない。だが，それならばなおのこと，もはや「過渡期」と呼ぶべきではないのだ。変化こそがこの世界の常態であり，決して「落ち着くまでの一時的な過渡期」などではないのだから。

ここさえ乗り越えれば，と思ったところで，その先に落ち着きどころなどありはしない。そこには，次の激変が待っているだけだ。

過渡期幻想に別れを告げ，パソコンとの蜜月を取り戻そう

かつて，私たちがパソコンにひかれた理由は，それがユーザーに「パソコンなしには実現できないなにか」をもたらしてくれたことだった。それは，必ずしも役に立つことではなかったし，またその実現には，ユーザー自身が相応の努力をしなければならなかった。だが

「これまではできなかった，あんなことができるようになるのではないか」

という可能性を感じているとき，パソコンユーザーは確かに未来を覗き見ていたはずだ。そして，自分が「やりたいこと」を実現するために，さまざまな工夫を凝らすとき，パソコンユーザーはそのときそこにあるマシンの中に，未来のひな形を作り出していたはずだ。

それが，なまじパソコンが役に立つものになり，それが市販のものによってわりあい容易に実現できるようになったことで，パソコンとユーザーの幸福な蜜月が遠のいていったように思う。それはそれで時代の必然であって，この流れに逆らおうとしても無駄なことだろう。けれども，パソコンユーザーはそれより少しいきすぎて，なにかしらモノを「手に入れる」ことでしか，パソコンの持つ有用性を実現できない，と思い込んでしまったのではないか。さらには，手に入れられる便利さが，そのモノの新しさや購入金額の多少によって測れるかのように信じてしまったのではないか。

だから，パソコンユーザーが「過渡期」の呪文から自由になり，パソコンとの幸福な関係を取り戻すためには，「過渡期」幻想のなかで失ったものを回復することが大切だと思うのだ。

その第1は「その機能や製品は，いったいなんのために必要なのか」という問いだろう。逆にいえば，新しいものだから，これを使っていないと遅れそうだから，という強迫観念を振り切って，自分には不要なものに「NO」ということだ。そして，そのためには，ユーザー自身がそもそも「パソコンを使ってなにをしたいのか」を確かめる必要がある。

第2には，既存の「モノ」が提供してくれる額面どおりの機能に依存するのではなく，それらを組み合わせてなにができるかを考えるようにすることだ。あるいは，新しいモノを手に入れることで物事を解決しようとする前に，まずは手持ちのマシンやソフトの範囲で工夫する術を探すことだ。

ハードウェアの組み合わせをちょっと工夫することで，意外なほど便利に使えることもある。アプリケーションソフトウェアの機能をいくつか組み合わせるだけで，思わぬ効果を上げられることもある。そして，もしどうしても必要に見合うだけの機能を持ったソフトウェアを見いだせなかったら，その先にはプログラミングという広大無辺な沃野があることも知っていてよいと思うのだ。

パソコンはこれまでも，そしてこれからも激しい変化の中にあり続けるだろうけれども，ユーザーが「パソコンを使ってできること」に思いをはせ，それに自らの努力で近づこうとすることによって，初めて開ける「可能性」の領域を，まだたくさん残しているはずだ。そして，そのように信じる人たちによって，この新〝Oh!X〟は送り出されている。筆者のひとりでしかない私がいうのはおこがましいのだが，読者の皆さんには，パソコンを巡って，まだまだ多様な可能性があるのだということを，この1冊から感じ取ってほしいと思う。

後日談　『原稿を読んでくれた某氏との会話。』

X：半年に一度新機種が出て，前のが使いものにならないくらい性能が上がるというのは，別に悪いことじゃないって気もするけど。

O：まぁ新陳代謝が激しいのは事実だし，別に悪いという気はないんですけどね。

X：少なくとも，5年くらい同じ性能でやられるのに比べればマシかなあ……と。

O：わはは。ただ，本当に「前のが使いものにならない」のかってのには文句いいたいわけですよ。

X：でも，ある程度パワーないと現実的じゃない処理もあるし。

O：もちろんそれはありますけど，ユーザーすべてが，実用的な必要に迫られてパソコン使っているわけじゃないと思いますし。

X：んー，マシンの性能や機能を気にするような人だと，単に知的好奇心みたいな部分が大きいだろうね。

O：ええ。で，それだったら，速度や機能に文句をいうだけじゃなくて，もっと違ったアプローチもあるんじゃないか，と思うわけで。

X：それもわかるんだけど，最近だと，昔以上に困難に思われてる部分もあるよね。誰でもできるわけじゃない，というか。

O：そうですね。実は私も「できない」側の人間なんで，あんまり偉そうなこといえないんですけど。ただ，できる範囲で構わないから，パソコンに対してそういう構え方を持っているといいと思うんですよ。

X：うん，気の持ち方だけでも，そういうものがあると違うだろうね。

STUDIO X ON AIR

FROM READERS TO THE EDITOR

本当に長い間お待たせしました。ついに復刊号をお届けできるようになりました。「なぜ，いきなり読者コーナー？」と思う方もいるでしょうが，Oh!X読者と一緒に作っていく本ですので，読者コーナーは欠かせません。ここでは，http://www.softbank.co.jp/publishing/ohx/での復刊告知とともに募集したメッセージを掲載しています。

◆復活すると聞いて，感激もひとしおです。モニターを2回やったこともありますし，モニターの途中で父が他界したりと，思い出はつきません。1日に数回ずつ長岡さんの絵のあるホームページを見ては，心待ちにしています。最終号で私の「さよなら，マイパソコンライフ」という投稿が載りました。68とともに，ホビーのパソコンは捨てたつもりでした。道具としてのパソコンを購入，いじってみましたが，正直感動もなにもありませんでした。まあ，時代に取り残された人間の戯言なのかもしれません。でも，小回りの利く，自分でちょいちょいと，必要なツールを制作できる手軽さと，その楽しさを感じえないというのは本当でした。結局はOSではなく，言語でもなく「小さな箱庭」がほしかったのだと，そういう記事を思い出し「私もそう思う」と感じるこの頃です。高速な，さまざまなおまけのついた，きらびやかなマシンを前にしても，気がつけばX68000のキーを叩いている自分に気がつき「やはり，このマシンと自分のホビーは心中するしかない」と思うこともしばしば。インターネットはWindowsでアクセスしていますけど。つかの間でもいいから，もう一度夢を。そう思う人は多いと思います。現在は投稿する力もないけれど，読者参加のSTUDIO Xみたいなところの投稿ができたらいいのにと思います。自分たちで参加して作っていく正当な雑誌が絶えて久しいですから。

湯澤 聡（35）埼玉県

◆廃刊になったあとも地下活動（？）を続け，ついに復活するあたりが，Oh!MZ/Oh!Xらしくてとてもいいです。この日がくるのを待ち望んでいた同志たちが，祝杯をあげていることでしょう。私は，現在はAT互換機＋Windowsという情けない（？）ユーザーで，SMCやHitBitを売っていた会社で働いてはいますが，昔はMZやX1のユーザーで，MZ/X1魂は忘れないつもりです。Oh!Xバンザイ！

泉 昭彦（28）東京都　FMV-DESKPOWER TE53，FMV-BIBLO NC13D

◆右を見ても左を見ても，同じ画面の機械ばかりの今日この頃。昔，少しばかり背伸びして，魅力あるパソコンを手に入れたときは本当に嬉しかった。どこを触れても新鮮で，どことなく愛嬌もあって電源を入れることさえ楽しかった。晩年，世間の流れに反抗し使い続けたそのパソコンも次第に行き場を失い，いまでは暗い押し入れの中。けれども，丁寧に整備され設置されたこのX68000は，再び火がともるのを待ち続けている。Oh!X，新しい誕生日をおめでとう！

赤坂まこと（28）福岡県　X68000 XVI改XX，PowerMacintosh7600/G3/275，PC

◆いやぁ，嬉しいじゃないですか。Oh!X復刊！　昨日引越しがありまして，X68000ACEとX68040turboは無事，新居に移動しました。一緒に過去のOh!MZやOh!Xも新居に移動できました。Oh!MZと私はこれで5回目の引越しになります。半年に一度くらいOh!MZの記事を読んで，当時の自分を思い返しています。現在，某S社の無限ワッカのサポートをしていたりするので，S社のパソコンとはまだまだ縁が切れませんが，サポートで入ってくる若い人がDOSのことをぜんぜん知らなくて，DOSの使い方を知っているだけで，ジジィ扱いされるので時代が変わったんだなぁ，とジジィっぽく回顧しています。半年に一度くらいOh!MZを読み返して昔を回顧するのはやっぱりジジィになったからなのか？　Oh!X復刊おめでとう！

皆川朋久（27）埼玉県　X68040turbo，X68000ACE HD，HITACHI PR210T

◆Windows環境にしてから3年目。ゲームにCGと，やってることはX68000をメインで使っていたころと変わってないです。なかでもゲームでは「ジオグラフシール」で対戦ゲームの味をしめ，今現在では家庭内LANを組んでDoom2に始まり，QuakeやX-Wing，そしてDiabloとゲームといえば対戦ゲーム志向に偏ってしまいました。プラットホームは変わっても楽しいことは同じ。X68専門誌のころも楽しませていただきましたが，そんな楽しいことをまたやってくれそうな新Oh!Xに期待します。

林 大輔（25）埼玉県　Pentium IIデュアルの自作NTマシン，X68030

◆すいません，X68kは使ったことないんですが……カウンタ，12345の地雷ふんじまいました。あぁっ，ゆるしてっ。でも，昔，友達が持ってて凄い羨ましかったです。

須澤 諭（25）東京都　PC-9821Xa/c10w

◆「STUDIO X」……なつかしい……。生きて再びこのタイトルを見ることができるとは思っていなかった。

松場 崇（29）奈良県　X68000ACE

◆復活おめでとうございます。2,500円は少し高いと思いますが，絶対に買いたいです。しかし，この時期になっていきなり復活とは思いませんでした。ヤラレターってなかんじです。もしかして新機種の発表があったりして。わくわく。もし，かりに新機種が出るとしたら，やっぱり買っちゃうんだろうなー。お金ないのになー。まったく罪作りなパソコンですよ，ペケロク君は。「復刊第1号」とわざわざ書いてあるので，「第2号」にも期待しています。それではすばらしい雑誌を作ってください。さいなら。

@KliRA（24）三重県　X68000 EXPERTII，Power Macintosh 7600

◆おかえりなさい。

宮沢清太（23）東京都　X68000PRO

◆Oh!X復刊おめでとうございます。これでやっと前号のデバッグ情報にありつけますね。え？　まさか載ってないのですか？　ところで，http://www.softbank.co.jp/publishing/ohx/を見ていると，「復刊第1号」と書いてありますね。ということは……今後も楽しみにしてます。

田村彰啓（23）兵庫県　PC-9821Cx13

◆今年に入ってから，2年ぶりにパソコンを再び触り始めました。書店へ行ってみると，あまり見るべきところのない情報系の雑誌がほとんどで，Oh!Xのような読者が参加できる雑誌がないのに驚きました。単なるノスタルジーではなく，本当に「私たち」に必要とされて復刊を決めた（U）さんをはじめとするOh!X関係者に感謝します。また「私たち」と一緒に誌面を作っていきましょう。

飯田伸一（24）宮城県　X68000PRO，Aptiva E27

◆復刊おめでとうございます。NGで見たときはガセかと思いました……廃刊じゃなくて休刊でしたものね。Oh!Xを読み始めたときは小学生だった私も大学生。Cが高いと福袋を買っていたりしたのに，いまでは，やれJavaだのIP-V6だの手形法だの（？）と殺伐とした日々を送っています。ワープロ，通信端末としての機能こそWindowsマシンに移行しましたが，曲作りのメインはいまだにZ-MUSIC＋SUPERでやってます（グラフィック系は学校のMac）。どんな記事が載るか，ホント，楽しみです。濃い記事のある雑誌が少ないいま，すっごい期待してます。

濱田研一（20）神奈川県　X6800SUPER+CZ-613D

◆まずはめでたい。いよいよ新たな伝説に向けて動き出すわけですね。満開さんも互換機出すことだし，こ

んなことは世界的に見ても例がないでしょう。あとは今まで以上に我々ユーザーががんばらないと。せっかく動きだした山をとめないために。

佐藤幸治(22)千葉県　X68030 PC-9821Xa7

◆祝「Oh!X」復活！　まさか復活するとは思っていませんでした。それだけに，嬉しさドン，さらに倍，ってなもんです〜♪　ちなみに，わが家のX68kはディスプレイをテレビとして使うときのリモコン代わりとして，しか活躍していません(爆)。これを機会に活躍の場が増えるのだろうか，いや，増えるに違いない。また，気合と愛のこもった内容の濃い記事を期待しています。

ころりん(24)東京都
X68000SUPER，Power Macintosh 7600/132

◆XVI電源がのついに入らなくなった。分解したところ，電源の電解コンデンサ2個が焼き切れていることが判明。その他，他の電解コンデンサから液漏れも発見。全部交換しないと故障が再発するかもしれないと思って，電源の電解コンデンサすべて(13個)を交換することに決めた。はたして我がXVIは無事復活できるのか！？

中村隆児(26)千葉県
X68000XVI，Libretto60

◆素浪人になったのが4年前。ささやかながら買い支えのつもりで始めた定期購読が1回の更新も迎えることなく郵便為替に化け，深夜のOS発売に関する騒動を横目で見た3年前。帰ってきたものが携えてくるのが，懐かしさだけでなく新たななにかであることを期待しています。「支える人々」を見る限り失望はないと思いますが。

新井幹男(28)神奈川県　X68000 PRO，X1turboZ，PC-9801ns，PC/AT互換機

◆あのOh!Xの復活。3年前Oh!Xが休刊，皮肉なことに5日後にWin95の発売……。ひとつの時代の終焉，そして当時自分の行ってきたことが否定されたような暗い気持ちになっていましたが……久々に燃えてきました。〝このアナベル・ガトーは，3年待ったのだ！〟の気分です。がんばってください。

安藤 晶(28)和歌山県　X68000，X68000XVI

◆Oh!X復活おめでとうございます。まさか復刊するとは思っていなかったので，とてもうれしいです。私もX68000の後継機種を作ろうと意気込んで＃に入ったまではよかったんですが，そこでまず現実の厳しさを知り，しかも仕事が忙しくなるにつれて家でX68kを触る機会も少なくなり，パソコンへの興味がだんだん薄れてきているところでした。復活したOh!Xでパソコンの面白さを再発見できたらいいなと思っています。

浪越孝宏(25)奈良県
X68000XVI，Mebius MN-5000D

◀だい

◀SHIMSOFT　広島県

◆復刊おめでとうございます。満開製作所の零式の話といい，一般メディアにX68000の話題が久しぶりに上ってうれしい限りです。まさに「夢を再び」という感じで，忘れかけていた大事ななにかを取り戻したような気がします。そうだ！　私はまだ，いまも，これからも，X68000ユーザーなんだ！　スタッフの皆様ありがとうございます。

もりかず(30)栃木県
元祖X68000，(DESKTOP& NOTE)

◆Oh!XがMOOKとはいえ，復刊と聞いて鳥肌が立っちゃいました。月刊Oh!Xが休刊になってから約3年。振り返ってみると，昔の自分から考えると，「よくここまでプログラムが書けるようになったなあ」と思います。(と，いっても実際，業務で作ったものの1/3はシェルスクリプトですが……)。考えてみると，OSがなんなのかわからない状態で出会ったX68000にひと目ぼれし，バイトをして貯めたお金でX68000を買い，それとともに買い始めていったOh!Xがいまの私の原点のような気がします。現在でも，アルゴリズムの参考で見ますし，なによりその思想が，いまの私に色濃く影響を与えているので。やはり，パーソナルコンピューティングという思想を忘れると，ブラックボックスを動かしているだけ，みたいな感じがしてやっぱり駄目ですね。でも，パーツを組み合わせて作っていく，手作りAT互換機もそれなりに(マシンの)ドライブ感はありますが，やっぱり，骨の髄までしゃぶることができるX680x0が私にはいちばんですね。こんな風に，いまだにX680x0から抜けられない私ですので，Oh!X復刊号だけじゃなく，2号，3号と続くことを願っています。ではでは。

菅野 誠(26)大阪府　X68030，X68000XVI，ThinkPad535E，PB100F，PC1350

◆最近のPCへの不満です。昔は必ずPCを購入すると，BASICなどの言語がついており，それだけでプログラミングの勉強ができました。X68000だって，X-BASICが付属してましたよね？　Windows全盛の現在では，市販アプリのプラットホームとしてPCはすばらしい発展を遂げてきましたが，プログラム開発の環境を整備するためには，さらなる出資が必要です。こんな状況でアイデア豊富なプログラマが育つとは，とても思えません。今も昔も，PCでプログラミングを勉強したいユーザーは大勢いると思います。OSにいろいろな機能をつけることを否定するわけではありませんが，そういうことをする以前に，たとえばWindowsならVBのサブセット版みたいなものでもいいから，OSに標準で用意してくれたらなぁとつくづく感じます。早い話，昔はヨカッタ……。

毛内俊行(31)神奈川県
自作DOS/V(GA-586HX2 K6-166MHz)

◆高校生の頃から愛読していた私も，いまではセガサターン(○イトー系移植が中心)やプレステなどのプログラムをするまでになりました。ゲーム業界に限ったことではなく，すべてにおいてマワリに流されてしまいそうになる今日この頃ですが，この「復活」は自分の中の初心，あるいは原点に戻るいいキッカケになると思っています。「STUDIO Z-MUSIC ver.3.0」に載っていたのはちょっとびっくりした〝kuny〟でした。

国井 稔(22)北海道　X68000 SUPER-HD

◆Oh!X復活おめでとうございます。ついに目覚める日がやってきたわけですが，みなさんお元気だったのでしょうか？　僕は元気です。ところで，X JAPANのファンの方で，このOh!Xを間違えて買われた方，お手元のスイッチを押してください，どうぞ。

幸 俊威(23)大阪府
X68000EXPERT(2?)＋Xellent30s

◆Oh!X復刊おめでとうございます。そうかあ，もう3年ぐらい経つんですねえ。休刊して復活した雑誌っていままで存在しましたでしょうか……。初の快挙なのかもしれませんね。しかしOh!Xは復刊するわ，満開製作所はX68000互換機を発表するわ，いまになってX68000周辺が急に慌ただしくなってきましたね。喜ばしいかぎりです。当時(いまでもやってますが)同人サークル「T&H PROJECTS」というのをやってまして，結局X68000用のソフトはほとんど全部載せてもらいました。途中から98，Winへ移行していきましたが，X68000用の新作はいまでもあきらめていません。みなさん，うちのプログラマへ励ましのメールを送っ

てやってください(jin@annie.ne.jp)。または誰か手伝ってください。Pメーカー2をいっしょに作ろう！　いまではメンバー全員社会人になり，メーカー/ソフトハウスへと散っていきましたが，X68000の世界で得たものは大きかったと思っています。実際，最先端の現場にいる人の何割がX68000ユーザーだったのかと思うと，いまでもX68000ユーザーであることを誇りに思えます。さすがに普段の仕事で使うことはなくなりましたが，いつでも使えるようになっていますよ。さて，現在仲間うちで「68 World Expo」開催を画策しています。計画が進み次第，各メディアでアナウンスしていきますので，そのときはご協力お願いします。

吉田 央(29)大阪府
X68000EXPERT，PC-9821La10/S8

◆まさか本当に復活するとは思ってませんでした。すごいです。初めてOH!MZに触れて11年，Oh!FMの読者だった後輩と「ああいうさりげなく難しいことをやる雑誌ってないもんかなぁ」などといってたら，まんまと出てしまうということで，うれしい限り。Oh!Xがなくなってから，私の知識量が増えなくて困ってました。いまでも役に立つことがあるOh!Xだけに，期待してます。最終号で年間モニタをやってたので，この号も感想はお送りしたいです。

浅野 憲(26)東京都
X68000Compact，PowerMac7600

◆Oh!Xが休刊になって，読むPC雑誌がなくなって非常に寂しい思いをしてきましたが，(この間にDOS/V magazine Startをはじめ，ほかのPC雑誌に手を出したけど)復刊，ありがとうございます(おめでとうかなぁ)。この当時は，これはやっぱり名前だけの休刊であって，実際には廃刊かと思っていましたが，本当に休刊だったんですね。ひとまず，復刊第1号だけのようですが，そんなことはいわずに，2号，3号と続くようになってくれると読者としては，しこたま嬉しいです。
p.s.　復刊にあわせたように，岡村直也さんの4コマ「マナブとケイコ」が，最終回を迎え，新シリーズが，スタートする予感が……。

松岡拓也(24)愛知県　Frontier神代 FBX-400，
FMV-5200D9M，X68000PRO

◆Oh!X復活おめでとうございます。ムックだけでなく月刊化……せめて隔月……季刊誌でも……。休刊から約3年たっての復活で制作者の方々も変わってしまっていると思いますが，昔ながらのOh!Xに期待しております。ちなみに，祝一平さんの大ファンなんですが，今回の制作には参加なされてないんですか？　いまなにをやっておられるのか気になります。

藤好政則(31)三重県
自作機PC/AT(PentiumII/400)，
東芝Dynabook SS，IBM PT110，
日立チャンドラー，オリベッティ クアデルノ

◆まさかOh!Xが復活するとは思っていなかったので，かなり驚きました(満怪^H開製作所がX68k互換機を作る予定という話も驚きましたが)。テレビ『ロスト・ユニバース』も，ネオOh!X発売日頃に最終回を迎えているでしょうか(最近になって見始めたのがちと無念)。ところで，週刊少年マガジンNo. 36・37『グランツーリスモを創った男たち』はご覧になったでしょうか(漫画では女性スタッフもいるように見えるけど……)。旧Oh!X『ハードコア3Dエクスタシー』で，延々と車の挙動をシミュレートしていたのは，そういうことだったのですかね。車に興味がなくて，数学，物理に弱かった私は(← 理系なのに情けない)，「興味のなかった記事」に「ハードコア3Dエクスタシー」を挙げたことがありました。すいません。作中，評判の悪かった前作が「期日に間にあいさえすればそれでいいと思っている宣伝サイドに強引にせかされて，中途半端なできになってしまった」，「製作日数が十分あればもっと売れてた」という話から，「いや違う，作った自分たちが満足できないゲームが，売れないのは当然」という話になるところなど，製品開発に携わる者として，身につまされる思いです。私は実はグランツーリスモのことをいまだに全然知らないのですが，4年間かけて，車の挙動をすべて計算した，完璧なゲームを開発させたというのはすごい！　こういうモノを作りたいものです……(丹さんが最後に泣いてる場面って本当？)。

西村武司(27)奈良県　X68030 + 060turbo，
X68000PROII + Xellent30PRO，PC-E550

◆初めて聞いたときは，〝絶滅寸前〟とまでいわれかねない，残り少ない68ユーザーから流れ始めた，「悲しいまでの希望的観測」だと思いました。新機種が出る！　……なんて噂が流れたこともありましたね？「これで，また68ユーザーとして生きていける」と，少なからず喜んだのも，いまではただの笑い話となってしまいました。「どうせ今回も，ぬか喜びになるに決まっている……」正直な話，期待はしていませんでした。でも違ったのですねっ！　現時点では，どのような本になるのかわかりませんが，あの人たちが，アイツらが，戻ってくるのですから，期待に応えてくれるはずです，確信しています！　「Oh!X」復活，おめでとうございます！　思い起こせば当時は，寝ても覚めても〝68〟でした。クロックアップもしました。マーキュリーユニット(ヲタク版)も，載せました。小さくてカクカクした動画の画質に，一喜一憂しました。「Oh!X」で1年間，読者モニターを務めさせていただいたこともありました。かまたさん率いる〝DoGA〟のコーナーのクイズで採用されて，プレゼントをいただいたこともありました。書き始めたらキリがないのでやめますが，書けといわれれば，いくらでも書けますよ。いまではメインマシンも，自作AT互換機になってしまいました。使用用途も，用意されたものを動かすだけの毎日です。「Oh!X」……，またあの〝D.D.I.Y〟精神に溢れた本が帰ってくると思うと，いまからワクワクしてきます。心なしかウチの「PRO」も，嬉しそうに見える……今日この頃です♪

吉岡洋明(25)埼玉県
X68000 PRO，自作AT互換機

◀KRBK 神奈川県

◀小川伸輔(びぜん@備前屋)

僕らの掲示板

★ディスクマガジン「兎団」特別復活号を発行します。ただいま配布，投稿希望者を募集中です。内容はHTMLファイルでメディアもCDとフロッピーを用意しているのでX68000ユーザーではない人も募集しています。為替500円分と返信用の宛名を書いたカード，使用OSと希望メディアを同封し下記まで送ってください。「準備号」をお送りします。詳しい内容はホームページで。

http://www.toyota.ne.jp/~tenja/usagidan/
〒050-0071 北海道室蘭市水元町57-8
明徳寮B-408 笹山尚誉

We want you!

新しく出発したOh!Xでは広く雑誌記事の執筆スタッフを募集しております。プロ/アマ、経験は問いません。本名での掲載が不可能な場合はご連絡くだされば考慮いたします。とにかく面白そうなことをやっているなと思ったらまずご連絡ください。なっちゃないなあと思ったらとにかくご連絡ください。よりよい誌面を構成していくために協力していただける方を広く募集中です。

ネットワークに接続されている方であれば在京の方でなくても可です。やる気があるけど接続環境がないという方も不可ではありませんが，現状では編集者ひとりですべての作業を行っているので，連絡を密に取ることは困難になっています。できるだけ簡便な連絡媒体を使用しないと全体の作業に支障をきたしますので，ご理解をお願いいたします。

現状では旧ライター陣に頼っている部分が大ですが，人手は圧倒的に足りません。Oh!Xは読者の皆様と一緒に作る雑誌です。積極的に参加してくださることをお願いいたします。

また，広く投稿プログラム，投稿原稿なども募集いたします。作品をお持ちの方はOh!X制作実行委員会までお送りください。

プログラムはちょっと……という方でもイラストが描ける，3DCGモデリングができる，作曲ができる，ハード工作ができるなどガテンな方，ぜひご参加ください。プログラマだけど会社つぶれた，マンガ家だけど仕事がない，同人ソフト作ったけど売れない……と悩みを抱えているなどという方も多少はご相談の余地があります。

その他，スタッフというかたち以外に，広く投稿作品を募集しております。プログラム，グラフィック，音楽などコンピュータを使った（無理に使わなくてもいいけど，なんとなく使ってたほうがいいかも）作品を募集しております。

また，STUDIO Xへのご意見，イラストなどの投稿も常時募集しております。なお，投稿作品は原則として返却いたしませんので，バックアップなどはあらかじめ取っておいてください。

スタッフ応募並びに投稿希望の方は，

氏名／住所／年齢／職業／連絡先電話番号／e-mailアドレス／インターネットへのアクセス環境の有無／使用機種などを明記のうえ，

〒103-8501
東京都中央区日本橋箱崎町24-1
ソフトバンク㈱
DOS/V magazine編集部内
Oh!X制作実行委員会

までご連絡ください。

その他，

http://www.softbank.co.jp/publishing/ohx/

なども参照してください。

愛読者プレゼント

Ⓐ ちびちびX

しゃかんきょりたもつ氏制作のX68030型キーホルダーです。現物に忠実にミニチュア化されたものです。http://www.geocities.co.jp/SiliconValley-PaloAlto/4247/ で通販も行われています。

このキーホルダーを20名様に。

Ⓒ 特製Tシャツ

ご存じの方はご存じのとおりですが，毎度毎度こんなものですみませんねぇ。といいつつ，完全ハンドメイドから脱却してそろそろもっとちゃんとしたTシャツなどにしたいと思っております。よって，写真のものとは異なる場合があります。とりあえず，10名様に。

Ⓑ スタジャン68

こちらもしゃかんきょりたもつ氏の制作によるもの。Xロゴの入ったスタジャンを3名様に。サイズはS，M，L，LL，3L，4Lがありますので，応募の際にサイズも指定してください（B-LLなど）。ひとサイズ大きめが推奨のようです。

Ⓓ グランツーリスモ

ポリフォニーデジタルさんからのいただきものです。気が向いたらサイン入りになるかもしれません。5名様に。

プレゼント応募の締め切りは1999年1月1日（当日消印有効）。当選者は，もし次の号がちゃんと出ましたら，そちらで発表いたします。もしくは発送をもって代えさせていただく，またはWebページ上で発表させていただくこともありますのでご了承ください。

編集室から from E・D・I・T・O・R

SIFT . BREAK

▶ここに発表したようなCGがメインのホームページを近々開設しようと思っています。よろしければ感想をお寄せください。参考にしたいと思いますので。
http://www.ecodacs2.nerima.tokyo.jp/~mori/ （森川）

▶初参加のさすらいライター、霧雨です。以後お見知りおきを。Oh!X復活、おめでとうございます。元（これからも）読者として、今後の発展を期待してます。今回の原稿ですが、Javaは仕様や状況が刻一刻と変化するので結構大変でした。これが出た時点ですでにJavaを取り巻く状況はさらに変わっているでしょうが、ゲームも作りやすい言語です。一度触ってみてください。 （霧）

▶7月末の締め切りを守った自分にご褒美（よく女性誌にはこんなコピーが書いてある）ってことでシャネルの夏期限定バッグを買いに行ったら、案の定品切れ。7月上旬発売のバッグだから当たり前っていえば当たり前だけど、がっかり。「締め切りを乗り越えたら代官山でデートだね」との甘～い約束は、原稿を入れたとたんにあっちが忙しくなっちゃうし。悲しくなってSONYモバイルパソコンのVAIO PCG-737/A4Gを買っちゃいました。ノートのくせしてPentium 233MMX搭載で、Windows 98を標準装備で、音がよくて、頭もよくて、添付ソフトはいっぱいあるし、筐体がおしゃれ～（自慢）。「ふっふっふ、これでホームページをじゃんじゃん更新できるぞ」と、野望がさらにふくらんだのでした。これはこれでいいんだけど、やっぱりシャネルのバッグと代官山でデートのほうがよかったよーな。 （Emi）

▶「フリッパーピンボール」を担当した市川幹人です。以前はMNM Softwareというソフトハウスを経営していたのでOh!Xを以前から読んでいた方には名前を覚えている方もいらっしゃると思います。1992年7月に過労から病にかかり1993年の4月に寝たきりになり、MNM Softwareを閉じました。36kgまで体重が落ちましたが1995年の2月になんとか完治し、現在ではマインドウェアというソフトハウスを営んでいます。今回はフリッパーの登場まで記しましたが、フリッパー初搭載のHumpty DumptyはなぜHumpty Dumptyと名付けられたのか考えてみてください。ビデオゲームとは一味違うネーミングに気がついていただければ幸いです。最後に、現在失われている「ホビーユース主体のパーソナルコンピューティングの復権を目指す雑誌」の復権に多大な情熱をそそがれ、私に今回の原稿を書く機会を与えてくださった（U）氏に感謝を送ります。（市）

次はもっといっぱい
描きたいと思います…
高橋哲史

Z
Z

▶変な本を作りましょうよ、というのが合言葉だった。このムックを企画・編集した植木さんが、まだThe Windows編集部にいた頃のことだから、いまから1年近く前になる。雑談レベルの話としてはずっと前からあったのだが、The Windowsが休刊になって最後の忘年会となってしまった1997年末、いつかはそういう「変な本」でお仕事させてくださいね、と話したのが、いわば遠い約束になっていた。こういういい方がよいか悪いかはちょっと迷うのだが、かつてはマニアのための同人誌に近い雰囲気を色濃く持っていたパソコン雑誌も、1980年代を通じて、パソコンが広く普及するにつれ、しだいに「普通の雑誌」になっていった。それはある意味で時代の必然であったし、洗練されたといえる一面もあるから、そのことが悪いというつもりはない。だが、同時に、往時のパソコン雑誌が持っていた、ある種の熱気のようなものは、かなり薄れてしまったことも間違いないと思っている。

こういうと、パソコンそのものを趣味とする人はまだ少なからずいるし、パソコン雑誌の中にだってマニアックな性格のものがあるではないか、と思うかもしれない。だが、それらのほとんどは、実際には新しい品物を追いかけることを中心としている。使いこなしのための情報はあっても、細々としたハウツーを紹介することが中心で、パソコンを主体的に使うことを語らなくなってしまっているのだ。

でも、編集者やライターも、そういう気持ちをなくしたわけではない。むしろ、そういう記事を書く場が少なくなってしまったことを、非常に残念に感じていることが多いと思う。そして、いまのパソコンユーザーの中にも、そのような場としての雑誌を懐かしみ、あるいは必要と感じてくれる人も、きっとそれなりにいるのではないか。だったら、いまのパソコン雑誌からなくなってしまった要素をテンコ盛りにした「変な」雑誌を作ってみたいよね、というのが、冒頭で述べた合言葉の真意だった。

とはいえ、正直なところ私は、それはあくまで「遠い約束」でしかない、と思っていた。そういう雑誌を欲してくれる人は確かにいるのだけれども、その数と、そして作り手が提供したいクオリティとを見比べたときに、価格がどのくらいになるか、といった条件の数々を考えた場合に、実現はかなり難しいと思ったからだ。だから、植木さんから「ようやく「変な本」を作れそうだ」というメールをもらったときには、本当に驚き、そしてうれしかった。

しかも、誌名は“Oh!X”だという。世の中でいったん「休刊」したものが復刊することはまずない、といわれているのだが、そのタブーを打ち破っての復刊である。かつてのXシリーズ専門誌ではないものの、通底する思いには共通するものがある。そして、Xという文字には、未知数というイメージが重なる。特定のなにかではなく、すべてになりうるもの。パソコン雑誌のsomething elseを目指すには、うってつけだといえるだろう。

そして、それは同時に、読者にとっても“X”である。昨今のパソコン雑誌に漠然としたもの足りなさを感じていた人には、きっとその「欠けているなにか」を満たすものになると思う。復刊号を手にしたときには、私も読者のひとりとして、そういうものを感じ取りたいと、いまからワクワクしている。 （OGA）

▶1986年11月号にtiny XEVIOUS for mz700を掲載していただいてから、はや12年。今日の私があるのはOh!MZ/Xのおかげです。その復刊号に原稿が書けるなんてラッキー……のはず。復活版Oh!Xの原稿のFuture BASIC（Mac）の原稿は没につぐ没。友人も原稿を書いて出したとのことで「没何回あった？」と聞くと「没はないで」とのこと。毎日没をくらったのは私だけか(^^?　おかげで、もう勢いだけで書いた文章なんでおかしなところがあって（U）さんに大量修正されているかも……。

しかしMacで手軽にプログラミングというのは難しい。Future BASICだと短いコードなんで、まだ楽なんですがCでやると非常に煩雑になってしまいます。半分試行錯誤で作られたマシンとOSだからしかたないのかもしれませんが。OSも頻繁に変更されるので半年前の本などは実情にあわなかったりいろいろ。BeOSのようにスマートにできればよいのですけど。

Future BASICの原稿は苦労したけど、JavaScriptのほうは結構楽。というのも最近作られたものだから言語的に素直というかシンプルといった感じです。スクリプトはシンプルに作ってありますが、実際のところNetscapeとExplorerでは実装が異なっていて動いたり動かなかったり。いちばん面倒なのは昔のバージョンでもうまく動くような実装をしないといけないという部分。それに拍車をかけるのがNetscapeのバグとExplorerの仕様書にあるけど実装されていない命令などなど。次の2号があったらネタに苦しみそう。

それにしても復活版Oh!Xを受け入れてくれる読者というのは、どのくらいいるんだろう。いまの時代いくつかのアプリケーションを使えば、ほとんど自分の望むことはできてしまう。おかげでプログラムを組む必要はほとんどなくて、よくてマクロ、スクリプトといったところ。このままだと、現在プログラムを作成できる世代がいなくなる、世代交代したら実質的に終わっているんじゃないかな。日本では仕様書を用意しアジア（海外）でプログラミングし国内/海外で販売するといった方式になるのか。発想という点からすると、ほとんどの場合米国に負けてしまいそうだし、なんか先行き怪しい。

あとWebも開設してますので、よろしく。URLはhttp://www.shiojiri.ne.jp/~openspc/です。

2号目は、はたしてあるのか？ （古旗）

▶「星の王子さま」を20年ぶりくらいで読みました。毎日イライラしてんので、の一てんきにすごしたいです。 （うい）

▶今回の原稿は、一度総没になって、全部書き直しました。最初の原稿は、「CGアニメ作品の脚本のひねりだし方」で、ちょっと初心者向きではなかったからです。ということで、私はすでに次号用の原稿の準備ができています。次回もよろしく。慈善事業が好きなDoGAも、この不況の中、いい加減にしないと資金が底をついてしまいそう。ということで、L1、L2を海外で販売する計画が進んでいます。もうかれば、その資金でまた慈善事業を…。でも、とりあえずL3をどうするかが問題だけど。制作に参加していたTVアニメが終わった。TVアニメ制作の裏舞台が見られて面白かった。

来年は劇場映画のCGを作りたいなぁ。　(かまた)

▶いまやゲームの世界というのは、以前にも増してライトな方向、コンシューマーゲーム機に流れている。乱暴にいってしまうと、世の中の時代そもそもがライトな方向性なわけだ。そういう意味で、ゲームセンターのアーケードゲームを象徴する縦シューなんてのは、結局は10年前の時代遅れの遊びなのかもしれない。▼そういうことを考えながら今回の縦シューの話は書いて見たが、もうちょっと具体的な話のほうがよかったかなぁ。いまのゲームを楽しむだけではなく、遊ぶことから、もっとゲームについて考えて、もっともっとゲームを知ってほしい。そんなつもりで書いたのだけども、ゲームについて遊ぶ以上の好奇心が湧いてくれたのなら、それだけでも満足だ。▼よく読む漫画の湾岸ミッドナイト（講談社・楠みちはる）に「一度陸に上がってしまった魚は……もう長く水の中を泳げない」という台詞が出てくる。最近は製品紹介とかの原稿を書くことが多い自分が、今回のような論調風の原稿で、改めて陸に上がった魚のようになっていることに直面する切なさは、ちょっと複雑な気分だ。言葉が滑り、紡ぎだせない苦しみ。だけど、これを機会にまた水の中に挑戦するのもいいかもしれない。また泳ぎだせば、少しずつ長く泳ぐこともできるはずだからだ。▼3年経って以前の人を含め、こういうメンバーが揃ってみると、ほかの皆が錚々たる本物というのを実感。やっぱりOh!Xってのは、凄い本だったのだなぁというのをいまさら確認してみたりして。大事なことは、自分と比べないことかな。そういうのって結構大事なことなんだよね。うんうん。　(八)

▶すっかり肌寒くなってきました。本当に今年の夏は短かったですねぇ。さて、今夏最大の思い出は初の野外コンサート参加。8月8日に静岡県浜松市にある渚園へサザンオールスターズのコンサートに行ってきました。正直いうと、あんまり曲知らなかったんですが、そんなの全然関係ないですね。久々にはじけてきました。コンサートはよかったのですが、どこにでもマナーの悪い人はいるものですね。自分の出したゴミくらいは捨てずに持ち帰らないとねぇ……。年末コンサートも参加したいなぁ。チケット取れればだけど。　(H.K.)

▶やベーなー、そろそろ始めなきゃなー。なんもやってねーなー、でもやる気になんねーなー。なんて思ってたところ、某映画のイベントに行って、林○めぐみの生歌を聞いたら、一気にスパークして1日で10ページ以上を書き上げてしまった。ありがとう、林○めぐみさん。めぐみ様って呼んでいいですか？　っていうか、呼ぶ。結果的に原稿はぜんぜん余裕だったけど。ところで、SD-X筐体はどうだった？　私の周りでの評判は上々。ひょっとして秋葉のどこかで展示してもらうことになるかもしれないから、実物を見たい人は探してみよう。その間私は古いマシンでがんばらなきゃいけないんだけど。　(I.K)

▶今回のような記事を書いていると、サンプルプログラムを作るための"プラットフォームを問わないリアルタイム3D環境"がないことに気づきます。そもそも私がふだんプログラミングをやっているPC UNIXで満足に使える環境がないのです。いちばん有望そうなのはOpenGLで、PCの3Dカードに対応したライブラリが出回りさえすればWindowsと（たぶんMacintoshとも）共通のプログラムが書けるのです。Linuxが走っているVoodoo搭載PCに限定すれば環境があることはあるのですが、まず限定しているというのがよくないですし、Voodooですから別画面、ちょっと癖もあります。パーフェクトな選択肢はないものでしょうか。　(丹)

▶ここ何年かのうちに、ずいぶん変な方向に進んじゃったこの世界。本誌の復活で少しでも昔の熱い思いがよみがえってくれれば嬉しいんですけどね。まぁ、ミックジャガーだってまだまだ熱く歌っているんだし年齢なんて関係ないさっ！

銀河にまたひとつ神話が生まれる

★初めての皆さんこんにちは。かつてOh!Xという雑誌が存在していたことすら知らずにこの本を手に取られた方もいると思います。日頃パソコンを使っていて感じているものとは違ったパソコンの世界が見えてくればこの本の意義があります。あなたが求めていたものに近いものが感じられれば幸いです。

また以前は機種別の専門誌という枠がありましたので、Oh!Xという本の存在は知っていても実際には読んだことがないという人も結構いるのではないかと思っています。そういった「初めてのOh!X」な方からの率直な意見を求めています。ぜひ、愛読者はがきをお送りください。

さてさて、復刊という珍しいことをやったわけですが、出版業界全体で見れば、休刊から復刊した雑誌というのもないわけではありません（ガロとか……）。パソコン誌に限れば初めてのことかもしれません。ただ、まだまだ不定期刊のムック状態ですし、第2刊目を危ぶむ声も聞かれます。まだまだ予断を許さない状況です。

一応、次回は年明けくらい（1月になるか2月になるかは未定）を目指して作業を進めていきたいと思います。

現状ではライター陣もほとんどが社会人ですし、月刊ベースでの発行はまずありえないでしょう。たとえ営業的にちゃんと軌道に乗ったとしても、作る側が無理ですから。さらなる挑戦に向けて、新たなる戦力を募集しています。興味のある方は編集室までご連絡ください。

★さて、お久しぶりの皆様。お待たせしました。新しいOh!Xはいかがでしょうか。あい変わらずだと思う人もいるでしょうし、全然違うと思う方もいるでしょう。作る側としてはまったく別のものにするつもりではいるのですが、基本部分はどうしても重なってきますので似た雰囲気の本になっていくと思います。

システム的には少し変えていきます。今回はおおっぴらにペンネームを解禁いたしました。あまり理由なくペンネームというのは無責任でもありますし、好きではないのですが、いろいろと事情のある方が増えている状況を鑑みて、掲載時には匿名ないしペンネームというものも認めるべきであるという結論に達しました。STUDIO Xでも今回はペンネームを通しましたが、こちらは特にマズイと判断できる理由が書き添えられていない場合は本名で掲載することにするかもしれません。なお、いずれにせよ、匿名のみの投稿は受け付けておりません。

今回は本当にいろいろな方々に協力していただきました。出自を明らかにはできませんが、あんな会社の人とあんな会社の人とあんな会社の人が一緒の本に原稿を書いているというのはかなり珍しいことなのかもしれません。本誌はオープンな場所でさまざまな人々が利害を超えて意見を交わせる場所になれたらよいと思っております。

現在の業界に足りないのはそういった場所と、議論のベースにできるハードウェア環境だと思っております。「新しい箱庭」を作るというのも目標のひとつですし、さらに次世代を見据えた視点で記事を展開していくことも重要でしょう。21.5世紀へ向けての基盤作りを目指していきます。

広く皆様からのご意見をお待ちしております。　(U)

●普通パソコンというと、ワープロを打ったり計算をしたり、いろいろな作業をサッサッとやってくれる「不思議な魔法の箱」のように思いますが、実は「プログラムという積み木」を素にして、それをいろいろに組みあわせて思い思いの作品が作れるキャンバスみたいなものなんですね。

私はどちらかというと古いタイプの絵描きで、最近のアーティストのように絵を描くのにパソコンを使ったりしませんが、自分の頭の中にあるイマジネーションに沿って、線や面、色、光、影などをひとつずつ紙に描き込んでいき、ひとつの作品に仕上げる……そういう過程は、実は「Oh!X」が目指すプログラム感覚にも通じるところがあるように感じます。

絵を描くという作業は物事の仕組みを解析することから始まります。特に、宇宙絵画などの場合には、単に物理的な仕組みだけではなく、天文学やその根底に流れる人類学や歴史学も吟味します。そのうえで、できあがった絵を判断するのは、それを見る人間の感覚ですから、それらの要素、シンボルのビジュアルな組み合わせが人間にどんな感情の流れをもたらすか、実際に目で見ていちばん強い組み合わせを探し出すにはコンピュータという便利なものが出てくる以前は、ひたすらたくさんのスケッチを描いてみて、途方もない時間と紙くずの山を経験に頼ったものです。

この場合、自分のスケッチの表現能力の限界にアイデアのほうが引きずられやすく、それから見たらほかの人が培ったプログラム資産を共有して利用できるということはものを作るうえでの革命ですね。そういう意味で、パソコンの原点に戻ってプログラムやゲームなどを手作り感覚で作ることを目指す「Oh!X」の登場には、私も大いに期待しています。頑張ってくださいね。　(長岡秀星)

で、私事ですが、いまのパソコン界があまりにつまらなくなっちゃったことに加えて、若干の私的な事情により、このたび会社を辞めて独立することにしました。気がついたら14年半も勤めていたわけで、まぁ我ながらよく続いたものだ。自宅がそのまま仕事場という、いまどき流行りの言葉でいえばSOHO。ハード関係の設計がメインだけど、そのうち周辺機器の製造/販売までいけたらいいなぁ。

http://www.teleway.ne.jp/~pastelmagic（開設はちょっと遅れるかも）

e-mail: pastelmagic@po.teleway.ne.jp　(Ku)

▶なんか、原稿、リテイクだらけだったみたいに書かれてますが、そんなでもないです。まぁ、大半に出しましたけど……。限定された条件でモノを作るのはなんだって同じことで、許された範囲でできるだけのことをするしかないのも同じです。いまも昔も変わりません。現時点でできるだけのことはしてみました。盛り込めなかったものも多いですが、それも実力のうちでしょう。今回は多くの方々のご協力をいただき、同時にご迷惑をおかけしました。無謀な試みを支えてくださったすべての方々に感謝いたします。　(U)

microOdyssey

■Oh!Xの休刊は意外と淡々と行われた。旧最終号をいま見返すと当時の状況が誌面からにじみ出してきている。Oh!XをOh!Xとして維持していくことは制作サイドから見て困難になった。「多少赤が出ても続けてはどうか」と、うちの社長にしては異例の発言も出たようだが、作れないものはしかたがない。形だけのものをOh!Xとして出していくことはできないというのはスタッフの共通した見解だった。

約10年ほどべったりとXシリーズのことだけを扱ってきたので、ほかの機種のこととなるとまるでわからない。先行きはわからないがとにかく編集部は片付けなければならない。社内でもっとも古い雑誌だっただけに十数年分の機材は大量で、私の管理分だけでも相当なものになる。1カ月くらいかけて、捨てるものは捨て、しかるべきものはしかるべきところに送り、どうしても捨てられないものはうちに送る。会社に置きっぱなしだった私物はできるだけ持ち帰る。ダンボール箱に山ほどの荷物を見て感慨にふける。楽器類、パソコン、周辺機器、書籍類、ゲーム機など生活の一部として存在していたものたちだ。最後に古い投稿袋を引っ張り出して分別してゴミ袋に叩き込む。創刊当時はともかく、私が入る前からの丸10年分くらいの全投稿作品がダンボールに入って保存されていたのだ。明け方の帰り道、茅場町へ向かう途中、投稿袋のことが思い出される。

整理しているとX1のツールなどが出てきて「ああ、これは載せてあげたかったなあ……」とか、ちょっとしたタイミングで載せそこねたり、ページが確保できなくて紹介できなかった投稿などが思い起こされ、思わず目頭が熱くなる。そして、そのときようやくOh!Xが終わったんだなあと感じた。

「私たちは応援団だ」

と、かつて@氏はいった。読者やユーザーのためにいろいろやってあげようとしても、全部を代わりにやってあげることはできないし、なにもかもできると思っちゃいけない。あくまでもそれをやるのは個々の読者であって、私たちはただ頑張れとしかいってあげられない。でも、それはとても大事なことなんだ、と。

それから9年、私なりのやり方でOh!MZ/Xを作り続けたが、結局のところ納得のいくかたちで結果は出せていない。9年間はさまざまな意味で準備期間だった。さまざまな環境を作り、実験を繰り返し、可能性を探ってきた。それでもまだまだやり残したことがある。それがようやく新しいOh!Xという形で再開できることになった。

NetXのページをご存じの方なら別に不思議には思わないのかもしれないが、少しずつ構想は固めつつ方向性を探っていた。DOS/V Start、The Windowsとも、現状のPCを勉強するにはよい媒体だった。しかし、古くからのライターとは連絡が取れなくなっており、状況はよくなっているわけではない。

「Oh!Xを作りたいんですが」

唐突に切り出してみた。皆、一様に驚くものの、なぜか誰も止めようとしない。

「全部で16折、ほとんど4色ページで、CD-ROMは2枚です」

なぜか誰も止めない。

「2折分はコート紙を使います。原稿料はこれくらいで定価はこれくらいで……」

なぜか誰も止めない。

「私ひとりでやります」

なんで誰も止めてくれないんだ、と、ほんの少し思った。　(U)

次回予告　◆1999年 春◆

特集　『ネットワークゲーム』

詳細 ………… 未定
定価 ………… 未定
参加スタッフ ………… 未定

Oh!X

■1998年11月1日発行　定価2500円(本体2381円)
■発行人　岡崎眞
■編集人　稲葉俊夫
■発売元　ソフトバンク株式会社
〒103-8501　東京都中央区日本橋箱崎町24-1
出版事業部
URL http://www.softbank.co.jp/publishing/ohx/
編集部　☎03-5642-8133 (DOS/V magazine編集部内)
販売局　☎03-6542-8100
■印刷　クニメディア

Oh!X 愛読者カード

郵便はがき

料金受取人払

日本橋局承認

3181

差出有効期間
平成12年5月
14日まで

103-8730

346

（受取人）
日本橋郵便局
私書箱358号

ソフトバンク(株)
DOS/V magazine 編集部 内
Oh!X 係行

フリガナ	年齢	性別
お名前	歳	男・女
ご住所 〒 TEL		
勤務先名（学校名）　職種 □技術 □営業 □事務 □その他	パソコン使用歴　年	
購読パソコン誌	プレゼント希望番号	

復刊第1号通巻168号

満開の電子ちゃん

作・え 岡村祭

あいかわらず電脳倶楽部は健在です! 振替口座が変わったので注意してね。

おお生き返ったぞ

よかったよかった

君らは冷凍睡眠機の事故にまきこまれて3年間眠ってたんだよ

3年?!

じゃ今は1998年…

電子!お母さんどんなに心配したか!

「もののけ姫」は?もののけ姫はもう公開されましたか

キミの最大関心事はそれかね

うわー WINDOWS98?!

え?消費税5%?

セ●のゲーム機がドリームキャストに!!

日本ワールドカップ出場…

負けちゃったけど

はあ…3年間て意外と大きいんだなあ

浦島太郎の心境ですよ

……誰か満開製作所に電話してみなよ

やめてよそんな死者にむち打つようなまね

なんてこと言うんですか

そうですよシーラカンスの例もあるしつぶれたと決まったわけじゃ

そうだあれ捜してみよう

満開製作所のホームページ

ホームページって…

いやそれ違う

それも違うっていうか「ページ」しか合ってない

その頃満開製作所

ジュワ～…♪

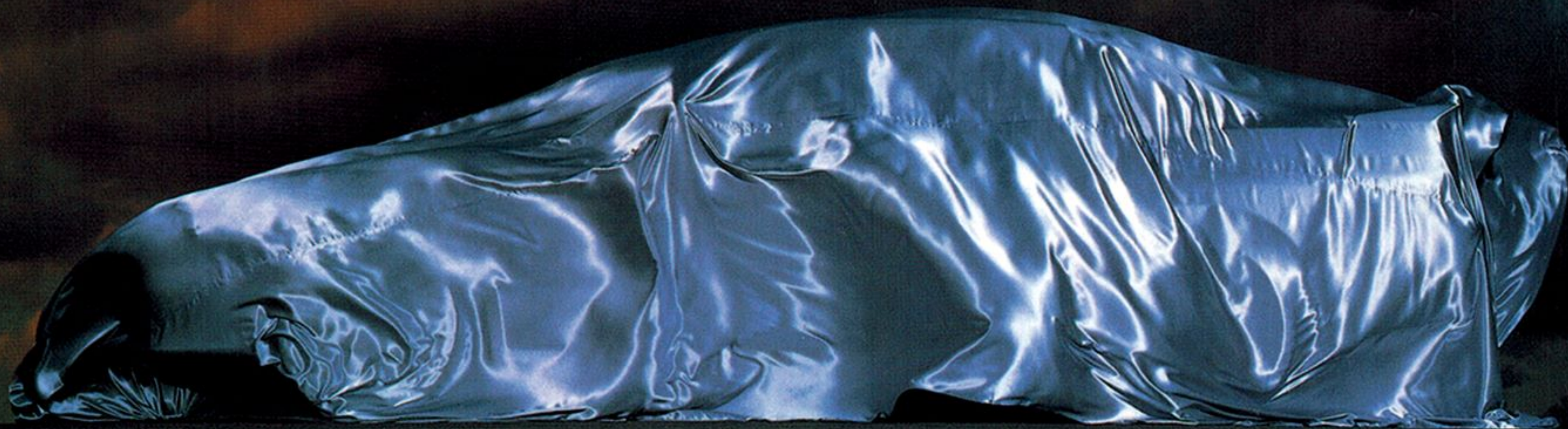
ハァ～♪CD ROMなら大容量

あっここんとこ少しこげてます

ぺったんぺったん

満開製作所Webサイトできました。 ホントです! http://www.mankai.co.jp/

コナン（東京都?）

人は私をドクターKと呼ぶ。私の神業とも言うべき治療により、また今日も、人々に幸せが訪れる。生死を分ける手術の時間、極度の緊張で疲れた心を癒してくれるのが電脳倶楽部・・・それは宛ら、心の治療師か?・・・そして私は明日も頑張れるのだ。なに?急患だと?患者はハードディスクドライブか。よかろう・・しゅゆちゅ・しゅゆちゅ・・・

SOFTBANK MOOK

平成10年11月1日発行
ソフトバンク株式会社発行

定価 本体2,381円 +税

9784797304466

1929455023817

Printed in Japan クニメディア

ISBN4-7973-0446-4 C9455 ¥2381E

雑誌 65814-36